

Final Project 1 Report

Group: B

Participants: Atera Alam & Priska Mohunsingh

Project Title: Optimizing Patient Care: A Hospital Readmissions Database

Milestone 1:

Dataset:

The first dataset is collected from kaggle and the second dataset is collected from centers for medicare and medicaid services.

Dataset 1 Link: <https://www.kaggle.com/datasets/dubradave/hospital-readmissions>

Dataset 2 Link: <https://data.cms.gov/provider-data/dataset/9n3s-kdb3>

Problem Definition:

In this project, we will build an operational database to collect and store some patient data when they are admitted into the hospital. The goal for this database is to store data that will help in determining the trend and likelihood of patient readmissions. Patient readmission in hospitals can occur due to a suboptimal treatment plan and lack of quality patient care. Collecting this data will further help in predictive analysis of the likelihood of patient readmission. Furthermore, it will provide insight on various other helpful analytical viewpoints, such as common factors for a successful treatment.

Objectives:

- Reduce hospital readmission rates by using predictive analytics to identify patients at high risk of unplanned readmissions.
- Improve operational efficiency by tracking KPIs like length of stay, bed availability, and resource allocation.
- Enhance patient outcomes by identifying key factors leading to successful treatments and lower readmission rates.
- Support decision-making for healthcare administrators by providing insights into treatment effectiveness, staffing needs, and cost management.

Business Context:

Hospital readmissions are a significant challenge in the healthcare industry, often leading to increased operational costs and penalties under healthcare regulations. Developing an internal operational database focused on hospital readmissions will help healthcare institutions manage patient data more efficiently, while also generating valuable insights for improving care delivery.

This project aims to:

- **Streamline data management:** Efficient data storage, modification, and retrieval.

Final Project 1 Report

- **Ensure HIPAA compliance:** The database will be designed with data privacy and security in mind, protecting sensitive patient information.
- **Provide real-time insights:** Operational teams will have access to data-driven insights to improve patient flow and treatment planning.

Technical Approach:

We aim to build the operational database using a relational database management system (DBMS), such as PostgreSQL. The database will be designed to handle both structured and semi-structured data, focusing on scalability and data security. The following are key technical components:

- **Database Schema Design:** A well-structured relational schema will be designed to store patient demographics, medical history, treatment details, and readmission data.
- **ETL Pipeline:** An ETL process will be established to collect and clean the data before it's loaded into the database.
- **Predictive Analytics:** Machine learning models, such as logistic regression and decision trees, will be applied to predict readmission likelihood based on patient data.
- **Data Privacy & Security:** HIPAA-compliant protocols will be implemented to ensure sensitive data is encrypted and securely stored.

Challenges and Solutions:

Some challenges include the possibility of inconsistent or incomplete data that can hinder analysis. We will implement data cleaning techniques to ensure the accuracy and reliability of the stored data.

As the database grows, ensuring fast query performance is essential. Thus, we will design the database with normalization in mind and use indexing for high-performance queries.

Project Analysis Overview:

We aim to analyze several different relationships within the data. Some examples include being able to create KPIs like tracking the percentage of unplanned readmission rates and treatment completions, length of stay optimization, bed availability, cost analysis, and resources available for staffing.

Data Collection:

Some of the key data points that will be collected and analyzed in the database include:

Patient Demographics: Age group, gender

Final Project 1 Report

Medical History: Number of outpatient, inpatient, and emergency room visits prior to hospitalization

Treatment Information: Labs requested, procedures performed, medications administered

Physician Details: Specialty, procedures performed

Readmission Data: Whether the patient was readmitted, and the factors leading to readmission

KPIs for Analysis: Readmission rates, length of stay, cost analysis, and staffing resources

a g e	time_i n_hos pital	n_lab_ proced ures	n_pr oced ures	n_me dicati ons	n_o utpa tient	n_in pati ent	n_e merg ency	medic al_spe cialty	di a g - 1	di a g - 2	di a g - 3	gluc ose_ test	A1 Ct est	ch an ge	diabe tes_ med	rea dmi tted

- "age" - age bracket of the patient
- "time_in_hospital" - days (from 1 to 14)
- "n_procedures" - number of procedures performed during the hospital stay
- "n_lab_procedures" - number of laboratory procedures performed during the hospital stay
- "n_medications" - number of medications administered during the hospital stay
- "n_outpatient" - number of outpatient visits in the year before a hospital stay
- "n_inpatient" - number of inpatient visits in the year before the hospital stay
- "n_emergency" - number of visits to the emergency room in the year before the hospital stay
- "medical_specialty" - the specialty of the admitting physician
- "diag_1" - primary diagnosis (Circulatory, Respiratory, Digestive, etc.)
- "diag_2" - secondary diagnosis
- "diag_3" - additional secondary diagnosis
- "glucose_test" - whether the glucose serum came out as high (> 200), normal, or not performed
- "A1Ctest" - whether the A1C level of the patient came out as high (> 7%), normal, or not performed
- "change" - whether there was a change in the diabetes medication ('yes' or 'no')
- "diabetes_med" - whether a diabetes medication was prescribed ('yes' or 'no')
- "readmitted" - if the patient was readmitted at the hospital ('yes' or 'no')

Static and Transactional Data:

Static Reference Data: Data that provides context to the transactional data and doesn't change as frequently. Below are the static data that we will be incorporating in our project.

Final Project 1 Report

- **Patient Information:** Patient ID, Demographics, Contact Information and Insurance information.
- **Healthcare Providers:** Provider ID, Provider, Name, Field of Specialty, Contact Information.
- **Hospital Departments:** Department ID, Name, Location within the hospital, Services.
- **Medication Information:** Medication ID, Name, Dosage Forms, Manufacturer

Transactional Data: Transactional data is dynamic data that changes from more frequently than static data. Either a transaction or an event triggers the updating of this data.

The list below contains the transaction data that our project will use.

Transactional Data (dynamic data capturing events or transactions):

- **Admission Attributes:**
 - i. Admission ID
 - ii. Patient ID (Foreign Key)
 - iii. Admission Date and Time
 - iv. Reason for Admission
 - v. Admitting Department ID (Foreign Key)
 - vi. Attending Provider ID (Foreign Key)
- **Discharge Attributes:**
 - vii. Discharge ID
 - viii. Admission ID (Foreign Key)
 - ix. Discharge Date and Time
 - x. Discharge Disposition (e.g., Home, Transferred)
 - xi. Follow-up Instructions
- **Medical Procedures Attributes:**
 - xii. Procedure Record ID
 - xiii. Admission ID (Foreign Key)
 - xiv. Procedure Code (Foreign Key)
 - xv. Procedure Date and Time
 - xvi. Performing Provider ID (Foreign Key)
 - xvii. Procedure Outcome
- **Medications Administered Attributes:**
 - xviii. Medication Administration ID
 - xix. Admission ID (Foreign Key)
 - xx. Medication ID (Foreign Key)
 - xxi. Dosage
 - xxii. Route of Administration
 - xxiii. Administration Date and Time
- **Laboratory Test Attributes:**
 - xxiv. Lab Test ID
 - xxv. Admission ID (Foreign Key)

Final Project 1 Report

- xxvi. Test Type
- xxvii. Test Code
- xxviii. Test Date and Time
- xxix. Results
- **Readmission Records Attributes:**
 - xxx. Readmission ID
 - xxxi. Original Admission ID (Foreign Key)
 - xxxii. Readmission Date
 - xxxiii. Readmission Reason
 - xxxiv. Associated Costs
- **Billing Records Attributes:**
 - xxxv. Billing ID
 - xxxvi. Admission ID (Foreign Key)
 - xxxvii. Total Charges
 - xxxviii. Insurance Provider ID (Foreign Key)
 - xxxix. Claim Status
 - xl. Patient Out-of-Pocket Expenses
- **Patient Feedback Attributes:**
 - xli. Feedback ID
 - xlii. Admission ID (Foreign Key)
 - xliii. Satisfaction Score
 - xliv. Comments
- **Staff Scheduling Attributes:**
 - xlv. Schedule ID
 - xlvi. Provider ID (Foreign Key)
 - xlvii. Shift Date
 - xlviii. Shift Start and End Times
 - xlix. Assigned Department ID (Foreign Key)
- **Equipment Usage Logs Attributes:**
 - i. Usage Log ID
 - ii. Equipment ID (Foreign Key)
 - iii. Admission ID (Foreign Key)
 - iv. Usage Start and End Times
 - iv. Maintenance Issues Reported

Stakeholder Impact:

This project will benefit multiple stakeholders. By providing real-time data on patient flow and treatment effectiveness, **administrators** can make more informed decisions on staffing, resource allocation, and patient care strategies. The system will provide insights into which treatments and procedures lead to fewer readmissions, helping **providers** improve the quality of care. A more efficient healthcare system ensures **patients** receive timely, effective care, reducing the risk of readmission and improving overall patient outcomes.

Final Project 1 Report

Milestone 2

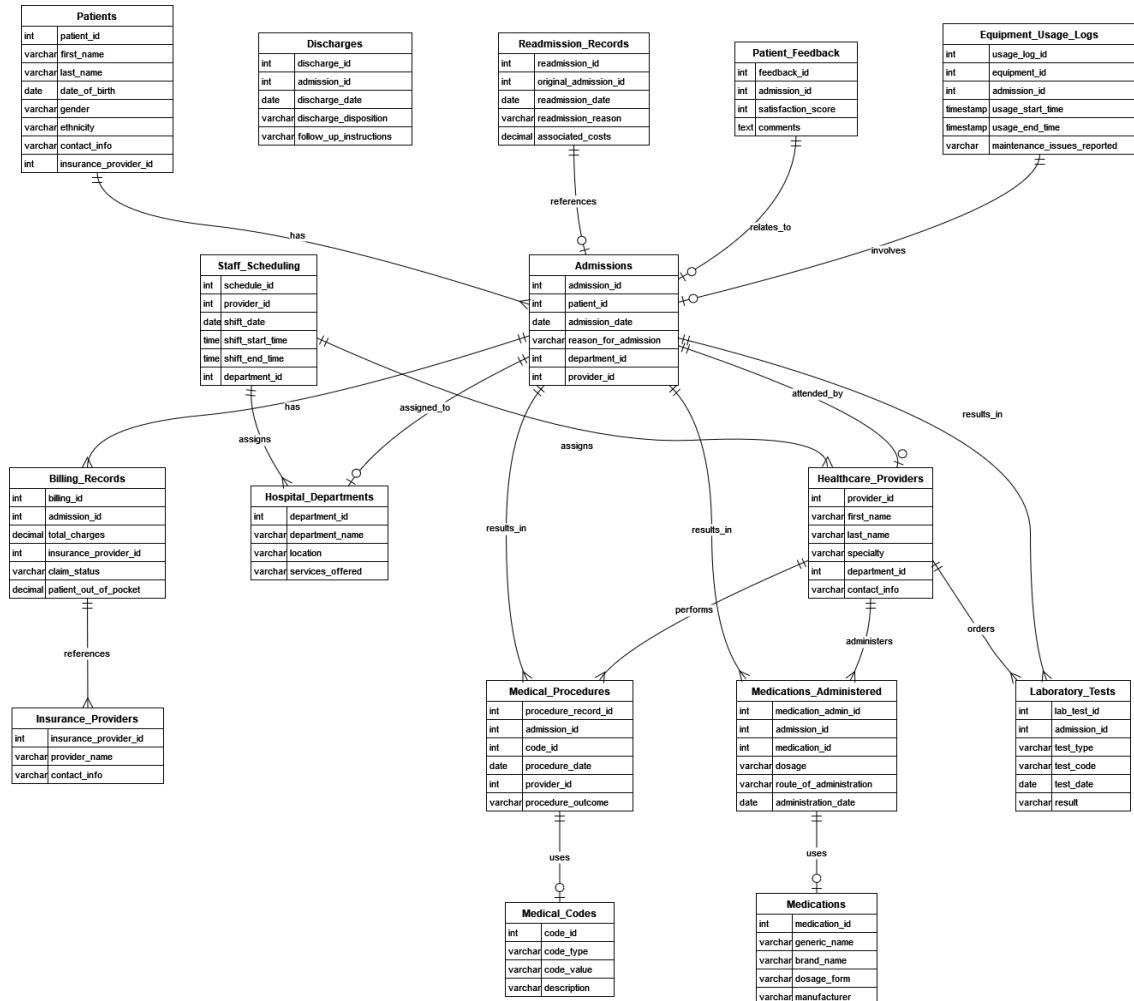
Business Context

Hospital readmissions present a significant challenge in healthcare, contributing to higher operational costs and potential penalties due to regulatory non-compliance. Our project aims to optimize patient care by developing an operational database that collects, stores, and analyzes patient readmission data. This database will assist healthcare administrators and clinicians in identifying key factors that lead to readmissions, thus improving patient outcomes, reducing costs, and enhancing operational efficiency.

The primary goal is to use this data for predictive analytics, identifying patients who are at a higher risk of unplanned readmissions and understanding the operational drivers behind successful treatments and reduced readmission rates.

Final Project 1 Report

Extended Entity-Relationship Diagram (ERD)



Static Reference Data Entities

- Patients
- Healthcare Providers
- Hospital Departments
- Medications
- Insurance Providers
- Medical Codes

Transactional Data Entities

- Admissions
- Discharges
- Medical Procedures
- Medications Administered
- Laboratory Tests
- Readmission Records
- Billing Records

Final Project 1 Report

- Patient Feedback
- Staff Scheduling
- Equipment Usage Logs

Entity Relationships

- Patients can have multiple Admissions.
- Admissions are associated with one Patient, one Healthcare Provider, and one Hospital Department.
- Admissions can lead to multiple Medical Procedures, Medications Administered, and Laboratory Tests.
- Healthcare Providers perform Medical Procedures, administer Medications, and order Laboratory Tests.
- Medications Administered reference Medications.
- Billing Records are linked to Admissions and Insurance Providers.
- Readmission Records are associated with previous Admissions.
- Patient Feedback is linked to Admissions.
- Staff Scheduling connects Healthcare Providers to Hospital Departments.
- Equipment Usage Logs link Equipment to Admissions.

Mapping ERD to Relational Model and Normalization

After creating the ERD, we mapped it to a relational model by defining tables for each entity and specifying relationships through primary and foreign keys.

Relational Model Tables

1. Patients

- patient_id (Primary Key)
- first_name
- last_name
- date_of_birth
- gender
- ethnicity
- contact_info
- insurance_provider_id (Foreign Key)

2. Healthcare Providers

- provider_id (Primary Key)
- first_name
- last_name
- specialty
- department_id (Foreign Key)
- contact_info

3. Hospital Departments

- department_id (Primary Key)

Final Project 1 Report

- department_name
- location
- services_offered

4. Medications

- medication_id (Primary Key)
- generic_name
- brand_name
- dosage_form
- manufacturer

5. Insurance Providers

- insurance_provider_id (Primary Key)
- provider_name
- contact_info

6. Medical Codes

- code_id (Primary Key)
- code_type (Diagnosis, Procedure)
- code_value
- description

7. Admissions

- admission_id (Primary Key)
- patient_id (Foreign Key)
- admission_date
- reason_for_admission
- department_id (Foreign Key)
- provider_id (Foreign Key)

8. Discharges

- discharge_id (Primary Key)
- admission_id (Foreign Key)
- discharge_date
- discharge_disposition
- follow_up_instructions

9. Medical Procedures

- procedure_record_id (Primary Key)
- admission_id (Foreign Key)
- code_id (Foreign Key)
- procedure_date
- provider_id (Foreign Key)
- procedure_outcome

10. Medications Administered

- medication_admin_id (Primary Key)
- admission_id (Foreign Key)
- medication_id (Foreign Key)
- dosage

Final Project 1 Report

- route_of_administration
- administration_date

11. Laboratory Tests

- lab_test_id (Primary Key)
- admission_id (Foreign Key)
- test_type
- test_code
- test_date
- result

12. Readmission Records

- readmission_id (Primary Key)
- original_admission_id (Foreign Key)
- readmission_date
- readmission_reason
- associated_costs

13. Billing Records

- billing_id (Primary Key)
- admission_id (Foreign Key)
- total_charges
- insurance_provider_id (Foreign Key)
- claim_status
- patient_out_of_pocket

14. Patient Feedback

- feedback_id (Primary Key)
- admission_id (Foreign Key)
- satisfaction_score
- comments

15. Staff Scheduling

- schedule_id (Primary Key)
- provider_id (Foreign Key)
- shift_date
- shift_start_time
- shift_end_time
- department_id (Foreign Key)

16. Equipment Usage Logs

- usage_log_id (Primary Key)
- equipment_id (Foreign Key)
- admission_id (Foreign Key)
- usage_start_time
- usage_end_time
- maintenance_issues_reported

Normalization

Final Project 1 Report

Tables then needed to be normalized to reduce redundancy:

- First Normal Form (1NF): All tables have primary keys, and all columns contain atomic values.
- Second Normal Form (2NF): All non-key attributes are fully functionally dependent on the primary key.
- Third Normal Form (3NF): No transitive dependencies; non-key attributes depend only on the primary key.

Implementing Relational Model in PostgreSQL

Two table creation examples below:

```
CREATE TABLE Patients (
    patient_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    gender VARCHAR(10),
    ethnicity VARCHAR(50),
    contact_info TEXT,
    insurance_provider_id INTEGER REFERENCES
InsuranceProviders(insurance_provider_id)
);
CREATE TABLE Admissions (
    admission_id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES Patients(patient_id),
    admission_date DATE,
    reason_for_admission VARCHAR(255),
    department_id INTEGER REFERENCES HospitalDepartments(department_id),
    provider_id INTEGER REFERENCES HealthcareProviders(provider_id)
);
```

Identifying Potential Dimensions, Hierarchies, and Measures

Dimensions

- Patient Dimension
 - a. Attributes: patient_id, age_group, gender, ethnicity, insurance_provider_id
 - b. Hierarchy: Age Group > Gender > Ethnicity
- Time Dimension
 - a. Attributes: date, day, month, quarter, year
 - b. Hierarchy: Day > Month > Quarter > Year
- Provider Dimension
 - a. Attributes: provider_id, specialty, department_id
 - b. Hierarchy: Provider > Specialty > Department

Final Project 1 Report

- Diagnosis Dimension
 - a. Attributes: code_id, code_value, description
 - b. Hierarchy: Code Value > Code Type (Diagnosis/Procedure)
- Geographic Dimension
 - a. If you have location data for patients or providers.
 - b. Hierarchy: City > State > Region

Measures

- Number of Readmissions
- Average Length of Stay
- Total Charges
- Patient Satisfaction Scores
- Medication Usage Counts

Hierarchies

- Organizational Hierarchy: Department > Division > Hospital
- Time Hierarchy: Hour > Day > Week > Month > Year

Slowly Changing Dimensions (SCDs)

- Patient Insurance Information: Could change over time (Type 2 SCD).
- Provider Roles or Departments: Providers may switch departments (Type 2 SCD).

```
import psycopg2
import pandas as pd
from faker import Faker
# Database connection parameters
db_params = {
    'host': 'localhost',
    'database': 'Project 1 - Healthcare Database',
    'user': 'user',
    'password': 'password'
}

# Connect to PostgreSQL database
conn = psycopg2.connect(**db_params)
cursor = conn.cursor()
```

Final Project 1 Report

```
# Initialize Faker for synthetic data generation
fake = Faker()

# Generate synthetic insurance providers
insurance_providers = []
for _ in range(10):
    provider = (
        fake.company(),
        fake.address()
    )
    insurance_providers.append(provider)

# Convert to DataFrame
insurance_df = pd.DataFrame(insurance_providers, columns=['provider_name',
'contact_info'])

# Insert data into InsuranceProviders table
for index, row in insurance_df.iterrows():
    cursor.execute(
        """
        INSERT INTO InsuranceProviders (provider_name, contact_info)
        VALUES (%s, %s)
        """,
        (row['provider_name'], row['contact_info'])
    )
conn.commit()
print("InsuranceProviders table populated successfully.")

departments = [
    ('Emergency', 'First Floor', 'Emergency care services'),
    ('Cardiology', 'Second Floor', 'Heart-related treatments'),
    ('Neurology', 'Third Floor', 'Brain and nervous system services'),
    ('Orthopedics', 'Second Floor', 'Bone and muscle treatments'),
    ('Oncology', 'Fourth Floor', 'Cancer treatments'),
    ('Pediatrics', 'First Floor', 'Child healthcare services')
]

# Define new departments
new_departments = [
    ('InternalMedicine', 'Third Floor', 'Internal medicine treatments'),
```

Final Project 1 Report

```
('Family/GeneralPractice', 'First Floor', 'General practice services'),
('Surgery', 'Second Floor', 'Surgical procedures and operations'),
('Emergency/Trauma', 'Ground Floor', 'Emergency and trauma care'),
('General Medicine', 'Second Floor', 'General medical services')
]

departments_df = pd.DataFrame(departments, columns=['department_name', 'location',
'services_offered'])

# Fetch department IDs from the database
cursor.execute("SELECT department_id, department_name FROM HospitalDepartments")
departments = cursor.fetchall()
department_mapping = {dept_name: dept_id for dept_id, dept_name in departments}

# Generate synthetic healthcare providers
providers = []
specialties = ['Emergency Medicine', 'Cardiology', 'Neurology', 'Orthopedics',
'Oncology', 'Pediatrics']
for _ in range(20):
    specialty = fake.random_choices(elements=specialties, length=1)[0]
    department_id = department_mapping.get(specialty.split()[0], None)
    provider = (
        fake.first_name(),
        fake.last_name(),
        specialty,
        department_id,
        fake.phone_number()
    )
    providers.append(provider)

providers_df = pd.DataFrame(providers, columns=['first_name', 'last_name',
'specialty', 'department_id', 'contact_info'])

# Insert data into HealthcareProviders table
for index, row in providers_df.iterrows():
    cursor.execute(
        """
        INSERT INTO HealthcareProviders (first_name, last_name, specialty,
        department_id, contact_info)
        VALUES (%s, %s, %s, %s, %s)
        """
        , (row['first_name'], row['last_name'], row['specialty'],
        row['department_id'], row['contact_info']))
```

Final Project 1 Report

```
        """
        (row['first_name'], row['last_name'], row['specialty'], row['department_id'],
row['contact_info'])
    )

conn.commit()
print("HealthcareProviders table populated successfully.")

medications = [
    ('Atorvastatin', 'Lipitor', 'Tablet', 'Pfizer'),
    ('Levothyroxine', 'Synthroid', 'Tablet', 'AbbVie'),
    ('Lisinopril', 'Prinivil', 'Tablet', 'Merck'),
    ('Metformin', 'Glucophage', 'Tablet', 'Bristol-Myers Squibb'),
    ('Amlodipine', 'Norvasc', 'Tablet', 'Pfizer'),
    ('Metoprolol', 'Lopressor', 'Tablet', 'Novartis')
]

medications_df = pd.DataFrame(medications, columns=['generic_name', 'brand_name',
'dosage_form', 'manufacturer'])

# Insert data into Medications table
for index, row in medications_df.iterrows():
    cursor.execute(
        """
        INSERT INTO Medications (generic_name, brand_name, dosage_form, manufacturer)
VALUES (%s, %s, %s, %s)
        """,
        (row['generic_name'], row['brand_name'], row['dosage_form'],
row['manufacturer'])
    )

conn.commit()
print("Medications table populated successfully.")

# Medical Code Generation and Integration

medical_codes = [
    ('Diagnosis', 'E11.9', 'Type 2 diabetes mellitus without complications'),
```

Final Project 1 Report

```
('Diagnosis', 'I10', 'Essential (primary) hypertension'),
('Procedure', '0W3P0ZZ', 'Insertion of Infusion Device into Subcutaneous Tissue'),
('Procedure', '3E0A3MZ', 'Introduction of Other Antineoplastic into Peripheral
Vein'),
]

medical_codes_df = pd.DataFrame(medical_codes, columns=['code_type', 'code_value',
'description'])

# Insert data into MedicalCodes table
for index, row in medical_codes_df.iterrows():
    cursor.execute(
        """
        INSERT INTO MedicalCodes (code_type, code_value, description)
        VALUES (%s, %s, %s)
        """,
        (row['code_type'], row['code_value'], row['description'])
    )

conn.commit()
print("MedicalCodes table populated successfully.")

## Static Patient Table
# Load Kaggle dataset
kaggle_df = pd.read_csv("/Users/ateraalam/Desktop/DATA ANALYTICS ENGINEERING/IE
6750/PROJECT_1_Datasets/hospital_readmissions.csv")

# Generate unique patient IDs
file_path = "/Users/ateraalam/Desktop/DATA ANALYTICS ENGINEERING/IE
6750/PROJECT_1_Datasets/hospital_readmissions.csv"

kaggle_df['patient_id'] = kaggle_df.index + 1

# Estimate date_of_birth based on age bracket
import pandas as pd
from datetime import datetime

def estimate_dob(age_range):
    # Clean up unwanted characters from the age range
```

Final Project 1 Report

```
age_range = age_range.replace('[', '').replace(']', '').replace(')', '')
''.replace('(', '').strip()
age_range = age_range.split('-')

if len(age_range) == 2:
    try:
        # Calculate the average age
        age = int((int(age_range[0].strip()) + int(age_range[1].strip())) / 2)
        # Estimate the year of birth
        birth_year = pd.Timestamp.now().year - age
        # Create a date object, assuming January 1st as the birth date
        date_of_birth = datetime(birth_year, 1, 1).date()
        return date_of_birth
    except ValueError:
        print(f"Invalid age range: {age_range}")
        return None
else:
    print(f"Unexpected format in age range: {age_range}")
    return None

kaggle_df['date_of_birth'] = kaggle_df['age'].apply(estimate_dob)

# Generate synthetic gender and contact_info
kaggle_df['gender'] = kaggle_df['patient_id'].apply(lambda x:
fake.random_element(elements=('Male', 'Female')))
kaggle_df['contact_info'] = kaggle_df['patient_id'].apply(lambda x: fake.address())

# Assign insurance_provider_id randomly
cursor.execute("SELECT insurance_provider_id FROM InsuranceProviders")
insurance_provider_ids = [row[0] for row in cursor.fetchall()]
kaggle_df['insurance_provider_id'] = kaggle_df['patient_id'].apply(lambda x:
fake.random_element(elements=insurance_provider_ids))

# Prepare patients DataFrame
patients_df = kaggle_df[['patient_id', 'date_of_birth', 'gender', 'contact_info',
'insurance_provider_id']].drop_duplicates(subset='patient_id')

for index, row in kaggle_df.iterrows():
    cursor.execute(
    """
    """)
```

Final Project 1 Report

```
INSERT INTO Patients (patient_id, date_of_birth, gender, contact_info,
insurance_provider_id)
VALUES (%s, %s, %s, %s, %s)
ON CONFLICT (patient_id) DO UPDATE
SET date_of_birth = EXCLUDED.date_of_birth,
    gender = EXCLUDED.gender,
    contact_info = EXCLUDED.contact_info,
    insurance_provider_id = EXCLUDED.insurance_provider_id
""",
(
    row['patient_id'],
    row['date_of_birth'],
    row['gender'],
    row['contact_info'],
    row['insurance_provider_id'],
)
)
)
conn.commit()
print("Patients table populated successfully.")

### Transactional Tables

# Assign random admission_date
start_date = datetime(2020, 1, 1)
end_date = datetime(2020, 12, 31)
kaggle_df['admission_date'] = kaggle_df['patient_id'].apply(
    lambda x: fake.date_between_dates(date_start=start_date, date_end=end_date)
)

# Map 'diag_1' to 'reason_for_admission' using MedicalCodes
cursor.execute("SELECT code_value, description FROM MedicalCodes WHERE code_type = 'Diagnosis'")
code_mapping = {row[0]: row[1] for row in cursor.fetchall()}
kaggle_df['reason_for_admission'] = kaggle_df['diag_1'].map(code_mapping)

# Fetch department IDs
cursor.execute("SELECT department_id, department_name FROM HospitalDepartments")
departments = cursor.fetchall()
department_mapping = {dept_name: dept_id for dept_id, dept_name in departments}
```

Final Project 1 Report

```
# Map 'medical_specialty' to 'department_id'
kaggle_df['department_id'] = kaggle_df['medical_specialty'].map(department_mapping)

# Fetch provider IDs
cursor.execute("SELECT provider_id, department_id FROM HealthcareProviders")
providers = cursor.fetchall()
providers_df = pd.DataFrame(providers, columns=['provider_id', 'department_id'])

# Function to assign provider_id based on department_id
def assign_provider(dept_id):
    available_providers = providers_df[providers_df['department_id'] == dept_id]['provider_id'].tolist()
    if available_providers:
        return fake.random_element(elements=available_providers)
    else:
        return None

kaggle_df['provider_id'] = kaggle_df['department_id'].apply(assign_provider)

# Prepare admissions DataFrame
admissions_df = kaggle_df[['patient_id', 'admission_date', 'reason_for_admission',
                           'department_id', 'provider_id']]
for index, row in admissions_df.iterrows():
    # Handle NaN or None values in the relevant columns
    reason_for_admission = row['reason_for_admission'] if pd.notnull(row['reason_for_admission']) else 'Unknown'
    department_id = row['department_id'] if pd.notnull(row['department_id']) else None
    provider_id = row['provider_id'] if pd.notnull(row['provider_id']) else None
    # Insert into Admissions, omitting admission_id
    cursor.execute(
        """
        INSERT INTO Admissions (patient_id, admission_date, reason_for_admission,
        department_id, provider_id)
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
            row['patient_id'],
            row['admission_date'],
            reason_for_admission,
            department_id,
```

Final Project 1 Report

```
        provider_id,
    )
)

conn.commit()

print("Admissions table populated successfully.")
# Fetch admission_id and patient_id from the Admissions table
cursor.execute("SELECT admission_id, patient_id FROM Admissions")
admissions_from_db = cursor.fetchall()

# Create a mapping between patient_id and admission_id
admission_id_mapping = {row[1]: row[0] for row in admissions_from_db}

# Map admission_id back to kaggle_df based on patient_id
kaggle_df['admission_id'] = kaggle_df['patient_id'].map(admission_id_mapping)

# Ensure default values for department_id and provider_id
kaggle_df['reason_for_admission'].fillna('Unknown', inplace=True)
kaggle_df['department_id'].fillna(1, inplace=True) # Assuming 1 as the default
department
kaggle_df['provider_id'].fillna(1, inplace=True) # Assuming 1 as the default
provider

# Update Admissions table with the correct values
for index, row in kaggle_df.iterrows():
    cursor.execute(
        """
        UPDATE Admissions
        SET reason_for_admission = %s,
            department_id = %s,
            provider_id = %s
        WHERE admission_id = %s
        """,
        (
            row['reason_for_admission'],
            row['department_id'],
            row['provider_id'],
            row['admission_id']
        )
    )
```

Final Project 1 Report

```
)\n\nconn.commit()\nprint("Admissions table updated successfully.")\n\n## Discharge Table\n\nfrom datetime import timedelta\n\n# Calculate discharge_date by adding 'time_in_hospital' days to 'admission_date'\nkaggle_df['discharge_date'] = kaggle_df.apply(\n    lambda row: row['admission_date'] + timedelta(days=row['time_in_hospital']), axis=1\n)\n\n# Generate 'discharge_disposition' and 'follow_up_instructions' synthetically\ndischarge_dispositions = ['Discharged to home', 'Transferred to another hospital',\n    'Left against medical advice']\nkaggle_df['discharge_disposition'] = kaggle_df['patient_id'].apply(\n    lambda x: fake.random_element(elements=discharge_dispositions)\n)\nkaggle_df['follow_up_instructions'] = 'Follow up with primary care physician in 2\nweeks.'\n\n# Prepare discharges DataFrame\ndischarges_df = kaggle_df[['discharge_date', 'discharge_disposition',\n    'follow_up_instructions']]\n\n# Fetch admission_ids from Admissions table\ncursor.execute("SELECT admission_id, patient_id, admission_date FROM Admissions")\nadmissions_records = cursor.fetchall()\nadmissions_df_db = pd.DataFrame(admissions_records, columns=['admission_id',\n    'patient_id', 'admission_date'])\n\n# Merge admission_ids back into discharges_df\ndischarges_df = admissions_df_db.merge(\n    discharges_df,\n    left_index=True,\n    right_index=True\n)
```

Final Project 1 Report

```
# Insert data into Discharges table without specifying discharge_id
for index, row in discharges_df.iterrows():
    cursor.execute(
        """
        INSERT INTO Discharges (admission_id, discharge_date, discharge_disposition,
follow_up_instructions)
        VALUES (%s, %s, %s, %s)
        ON CONFLICT (admission_id) DO NOTHING
        """,
        (
            int(row['admission_id']),
            row['discharge_date'],
            row['discharge_disposition'],
            row['follow_up_instructions']
        )
    )

conn.commit()
print("Discharges table populated successfully.")

import pandas as pd
from datetime import timedelta

# Prepare procedure records
procedure_records = []

# For each admission, I created procedure records based on 'diag_1', 'diag_2',
'diag_3'
for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    provider_id = row['provider_id']
    procedure_date = row['admission_date'] + timedelta(days=1)  # Assume procedure is
done the next day after admission
    procedure_outcome = 'Successful'  # Assuming all procedures are successful

    # Iterate over diagnosis columns to assign procedure codes
    for diag_col in ['diag_1', 'diag_2', 'diag_3']:
        code_value = row[diag_col]
        if pd.notnull(code_value):
```

Final Project 1 Report

```
cursor.execute(
    """
    SELECT code_id FROM MedicalCodes WHERE code_value = %s
    """,
    (code_value,)
)
result = cursor.fetchone()

if result is not None:
    code_id = result[0]
    procedure_records.append({
        'admission_id': admission_id,
        'provider_id': provider_id,
        'procedure_date': procedure_date,
        'code_id': code_id,
        'procedure_outcome': procedure_outcome
    })
else:
    print(f"Warning: No code_id found for diagnosis {code_value}")

# Convert the records into a DataFrame
procedures_df = pd.DataFrame(procedure_records)

# Insert data into MedicalProcedures table
for index, row in procedures_df.iterrows():
    # Skip the row if provider_id is NaN
    if pd.isna(row['provider_id']):
        print(f"Skipping row {index} due to missing provider_id.")
        continue

    cursor.execute(
        """
        INSERT INTO MedicalProcedures (admission_id, provider_id, procedure_date,
code_id, procedure_outcome)
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
            int(row['admission_id']),
            int(row['provider_id']), # Ensure it's a valid integer
            row['procedure_date'],
            code_id,
            row['procedure_outcome']
        )
    )
```

Final Project 1 Report

```
        int(row['code_id']),
        row['procedure_outcome']
    )

)

conn.commit()

print("MedicalProcedures table populated successfully.")

# Generate and insert Medications Administered data

medications_administered_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    medication_id = fake.random_int(min=1, max=6) # Assuming there are 6 medications
    dosage = f"{fake.random_int(min=1, max=2)} tablet(s)" # Generate dosage as 1 or 2 tablets
    route_of_administration = 'Oral' # Assuming oral route for simplicity
    administration_date = row['admission_date'] + timedelta(days=1) # Day after admission

    medications_administered_records.append({
        'admission_id': admission_id,
        'medication_id': medication_id,
        'dosage': dosage,
        'route_of_administration': route_of_administration,
        'administration_date': administration_date
    })

# Insert into MedicationsAdministered table
for record in medications_administered_records:
    cursor.execute(
        """
        INSERT INTO MedicationsAdministered (admission_id, medication_id, dosage,
        route_of_administration, administration_date)
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
            record['admission_id'],
            record['medication_id'],
            record['dosage'],
            record['route_of_administration'],
            record['administration_date']
        )
    )
```

Final Project 1 Report

```
        record['admission_id'],
        record['medication_id'],
        record['dosage'],
        record['route_of_administration'],
        record['administration_date']
    )
)

conn.commit()
print("MedicationsAdministered table populated successfully.")

# Generate and insert Laboratory Tests data

lab_tests_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    test_type = fake.random_element(elements=('Glucose', 'A1C', 'Blood Panel', 'Lipid Panel'))
    test_code = fake.random_int(min=1000, max=9999) # Assuming test codes are random integers
    test_date = row['admission_date'] + timedelta(days=2) # Lab test done 2 days after admission
    result = fake.random_element(elements=('Normal', 'Abnormal'))

    lab_tests_records.append({
        'admission_id': admission_id,
        'test_type': test_type,
        'test_code': test_code,
        'test_date': test_date,
        'result': result
    })

# Insert into LaboratoryTests table
for record in lab_tests_records:
    cursor.execute(
        """
        INSERT INTO LaboratoryTests (admission_id, test_type, test_code, test_date,
        result)
        """
    )
```

Final Project 1 Report

```
VALUES (%s, %s, %s, %s, %s)
"""
(
    record['admission_id'],
    record['test_type'],
    record['test_code'],
    record['test_date'],
    record['result']
)
)

conn.commit()
print("LaboratoryTests table populated successfully.")

# Generate and insert Readmission Records data

readmission_records = []

for index, row in kaggle_df.iterrows():
    original_admission_id = row['admission_id']
    readmission_date = row['admission_date'] + timedelta(days=fake.random_int(min=30,
max=100)) # Readmission happens 30-100 days later
    readmission_reason = fake.sentence(nb_words=6) # Random sentence for readmission reason
    associated_costs = fake.random_int(min=500, max=5000)

    readmission_records.append({
        'original_admission_id': original_admission_id,
        'readmission_date': readmission_date,
        'readmission_reason': readmission_reason,
        'associated_costs': associated_costs
    })

# Insert into ReadmissionRecords table
for record in readmission_records:
    cursor.execute(
        """
        INSERT INTO ReadmissionRecords (original_admission_id, readmission_date,
readmission_reason, associated_costs)
        VALUES (%s, %s, %s, %s)
        """
    )
```

Final Project 1 Report

```
"""
(
    record['original_admission_id'],
    record['readmission_date'],
    record['readmission_reason'],
    record['associated_costs']
)
)

conn.commit()
print("ReadmissionRecords table populated successfully.")

# Generate and insert Billing Records data

billing_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    total_charges = fake.random_int(min=5000, max=20000) # Random total charge between
    5000 and 20000
    claim_status = fake.random_element(elements=('Pending', 'Approved', 'Denied'))
    patient_out_of_pocket = fake.random_int(min=500, max=3000)

    billing_records.append({
        'admission_id': admission_id,
        'total_charges': total_charges,
        'claim_status': claim_status,
        'patient_out_of_pocket': patient_out_of_pocket
    })

# Insert into BillingRecords table
for record in billing_records:
    cursor.execute(
        """
        INSERT INTO BillingRecords (admission_id, total_charges, claim_status,
        patient_out_of_pocket)
        VALUES (%s, %s, %s, %s)
        """,
        (
            record['admission_id'],
            record['total_charges'],
            record['claim_status'],
            record['patient_out_of_pocket']
        )
    )
```

Final Project 1 Report

```
        record['total_charges'],
        record['claim_status'],
        record['patient_out_of_pocket']
    )
)

conn.commit()
print("BillingRecords table populated successfully.")

# Generate and insert Patient Feedback data

feedback_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    satisfaction_score = fake.random_int(min=1, max=5)  # Satisfaction score from 1 to
5
    comments = fake.sentence(nb_words=10)  # Random comment

    feedback_records.append({
        'admission_id': admission_id,
        'satisfaction_score': satisfaction_score,
        'comments': comments
    })

# Insert into PatientFeedback table
for record in feedback_records:
    cursor.execute(
        """
        INSERT INTO PatientFeedback (admission_id, satisfaction_score, comments)
        VALUES (%s, %s, %s)
        """,
        (
            record['admission_id'],
            record['satisfaction_score'],
            record['comments']
        )
    )
```

Final Project 1 Report

```
conn.commit()
print("PatientFeedback table populated successfully.")

# Generate and insert Staff Scheduling data

staff_scheduling_records = []

for index, row in providers_df.iterrows():
    provider_id = row['provider_id']
    shift_date = fake.date_between(start_date='-30d', end_date='today') # Last 30 days
    shift_start_time = fake.time()
    shift_end_time = fake.time()
    department_id = row['department_id']

    staff_scheduling_records.append({
        'provider_id': provider_id,
        'shift_date': shift_date,
        'shift_start_time': shift_start_time,
        'shift_end_time': shift_end_time,
        'department_id': department_id
    })

# Insert into StaffScheduling table

from datetime import datetime

staff_scheduling_records = []

for index, row in providers_df.iterrows():
    provider_id = int(row['provider_id'])
    department_id = row['department_id']

    # Skip if department_id is NaN
    if pd.isna(department_id):
        continue

    department_id = int(department_id)

    shift_date = fake.date_between(start_date='-30d', end_date='today') # Last 30 days
```

Final Project 1 Report

```
# Generate times and ensure proper format
shift_start_time_str = fake.time()
shift_end_time_str = fake.time()

shift_start_time = datetime.strptime(shift_start_time_str, '%H:%M:%S').time()
shift_end_time = datetime.strptime(shift_end_time_str, '%H:%M:%S').time()

staff_scheduling_records.append({
    'provider_id': provider_id,
    'shift_date': shift_date,
    'shift_start_time': shift_start_time,
    'shift_end_time': shift_end_time,
    'department_id': department_id
})

for record in staff_scheduling_records:
    try:
        cursor.execute(
            """
                INSERT INTO StaffScheduling (provider_id, shift_date, shift_start_time,
shift_end_time, department_id)
                VALUES (%s, %s, %s, %s, %s)
            """,
            (
                record['provider_id'],
                record['shift_date'],
                record['shift_start_time'],
                record['shift_end_time'],
                record['department_id']
            )
        )
        conn.commit()
    except Exception as e:
        print(f"Error inserting StaffScheduling record: {record}")
        print(f"Exception: {e}")
        conn.rollback()
print("Staff Scheduling table populated successfully.")

#equipment table
```

Final Project 1 Report

```
equipment_items = []

for equipment_id in range(1, 11): # Equipment IDs from 1 to 10
    equipment_name = f"Equipment {equipment_id}"
    equipment_type = fake.random_element(elements=('MRI Machine', 'CT Scanner', 'X-Ray
Machine', 'Ultrasound', 'ECG Machine'))
    maintenance_schedule = fake.date_between(start_date='today', end_date='+30d')
    department_id = fake.random_int(min=1, max=10) # Assuming department IDs from 1 to
10

    equipment_items.append({
        'equipment_id': equipment_id,
        'equipment_name': equipment_name,
        'equipment_type': equipment_type,
        'maintenance_schedule': maintenance_schedule,
        'department_id': department_id
    })

# Insert into Equipment table
for item in equipment_items:
    cursor.execute(
        """
        SELECT COUNT(*) FROM Equipment WHERE equipment_id = %s
        """,
        (item['equipment_id'],)
    )
    count = cursor.fetchone()[0]

    if count == 0: # Only insert if the equipment_id doesn't already exist
        cursor.execute(
            """
            INSERT INTO Equipment (equipment_id, equipment_name, equipment_type,
maintenance_schedule, department_id)
            VALUES (%s, %s, %s, %s, %s)
            """,
            (
                item['equipment_id'],
                item['equipment_name'],
                item['equipment_type'],
                item['maintenance_schedule'],
                item['department_id']
            )
        )
```

Final Project 1 Report

```
        item['maintenance_schedule'],
        item['department_id']
    )
)
else:
    print(f"Equipment ID {item['equipment_id']} already exists. Skipping
insertion.")
conn.commit()
print("Equipment table populated successfully.")
```

Milestone 3:

Multidimensional Model for Hospital Readmissions Data Warehouse

The multidimensional model for the hospital readmissions data warehouse is structured using a star schema, which is particularly well-suited for analytical queries in OLAP systems. The model is designed to support the analysis of hospital readmissions data, with the goal of improving patient care by identifying trends and factors that contribute to readmissions.

- Fact Table:
 - The Readmission Facts table serves as the core of the model, storing key performance metrics such as total charges, length of stay, satisfaction scores, and readmission rates. This table links to several dimensions, enabling comprehensive analysis of patient readmission patterns.
- Dimension Tables:
 - Patient Dimension: Contains demographic information such as age group, gender, ethnicity, and insurance provider details, allowing analysis based on patient characteristics.
 - Time Dimension: Facilitates temporal analysis with attributes like day, month, quarter, and year.
 - Provider Dimension: Captures details about healthcare providers, including their specialty and associated department.
 - Department Dimension: represents hospital departments, including their location within the facility.
 - Procedure Dimension: Stores medical procedure codes and descriptions to enable analysis of the types of treatments associated with admissions.
 - Insurance Dimension: Holds information about the insurance providers involved in patient care.

This multidimensional model allows for complex OLAP operations, such as slicing data by patient demographics, dicing across time and departments, performing roll-ups and drill-downs

Final Project 1 Report

through time hierarchies, and pivoting data to compare performance across departments or providers. These capabilities are critical for identifying patterns that can inform strategies to reduce hospital readmissions and enhance patient outcomes.

This fact table is designed to support key performance indicators and analyses by tracking the following measures:

Measures:

- Number of Readmissions
- Total Charges
- Satisfaction Scores
- Associated Costs

Cardinality:

- A patient can have multiple readmissions.
- A provider can perform multiple procedures during an admission.
- The time dimension is linked to many admissions and discharges, supporting historical and trend-based analysis.

This multidimensional model, structured with these fact and dimension tables, allows for in-depth analysis of hospital readmissions, enabling stakeholders to identify patterns and trends that can help improve patient outcomes and reduce operational costs.

Conceptual Model for Hospital Readmissions Data Warehouse

Entities and Relationships

- Core Entities:
 - a. Patient: Represents individuals admitted to the hospital.
 - b. Provider: Represents healthcare providers responsible for patient care.
 - c. Time: Represents dates and time periods related to admissions and discharges.
 - d. Diagnosis: Represents the primary medical conditions associated with readmissions.
 - e. Location: Represents location-related information for patient and provider distribution analysis.
 - f. Insurance Provider: Represents the patient's insurance provider, impacting cost and readmission trends.
 - g. Department: Represents hospital departments where care is provided, e.g., cardiology, emergency.
 - h. Readmission Facts: The central entity capturing each readmission event, aggregating information from all relevant dimensions.
 - i. Admission Facts: Another central entity capturing each admission event, aggregating information from all relevant dimensions.
- Relationships:

Final Project 1 Report

- a. A Patient is admitted by a Provider to a specific Department and has Insurance coverage.
- b. A Diagnosis is assigned to each Readmission Fact event, which may include associated Medications.
- c. Readmission Facts are time-stamped by Time dimensions to allow tracking over specific periods.
- d. Geographic dimensions relate to both Patient and Provider for location-based analytics.

Key attributes for each entity:

- Patient: Age group, gender, ethnicity.
- Provider: Provider name, specialty, department affiliation.
- Time: Date ID, day, month, quarter, year.
- Diagnosis: Code value, description.
- Geographic: City, state, region.
- Insurance Provider: Provider name, contact information.
- Department: Department name, location.
- Medication: Medication name, dosage form, administration route.
- Readmission Facts: Total charges, average length of stay, satisfaction score, associated costs.

Hierarchies

- Time: Supports drill-down through Day > Month > Quarter > Year for trends and seasonal analysis.
- Patient: Facilitates segmentation through Age Group > Gender > Ethnicity for demographic insights.
- Location: Enables regional analysis through City > State > Region to assess geographic patterns.

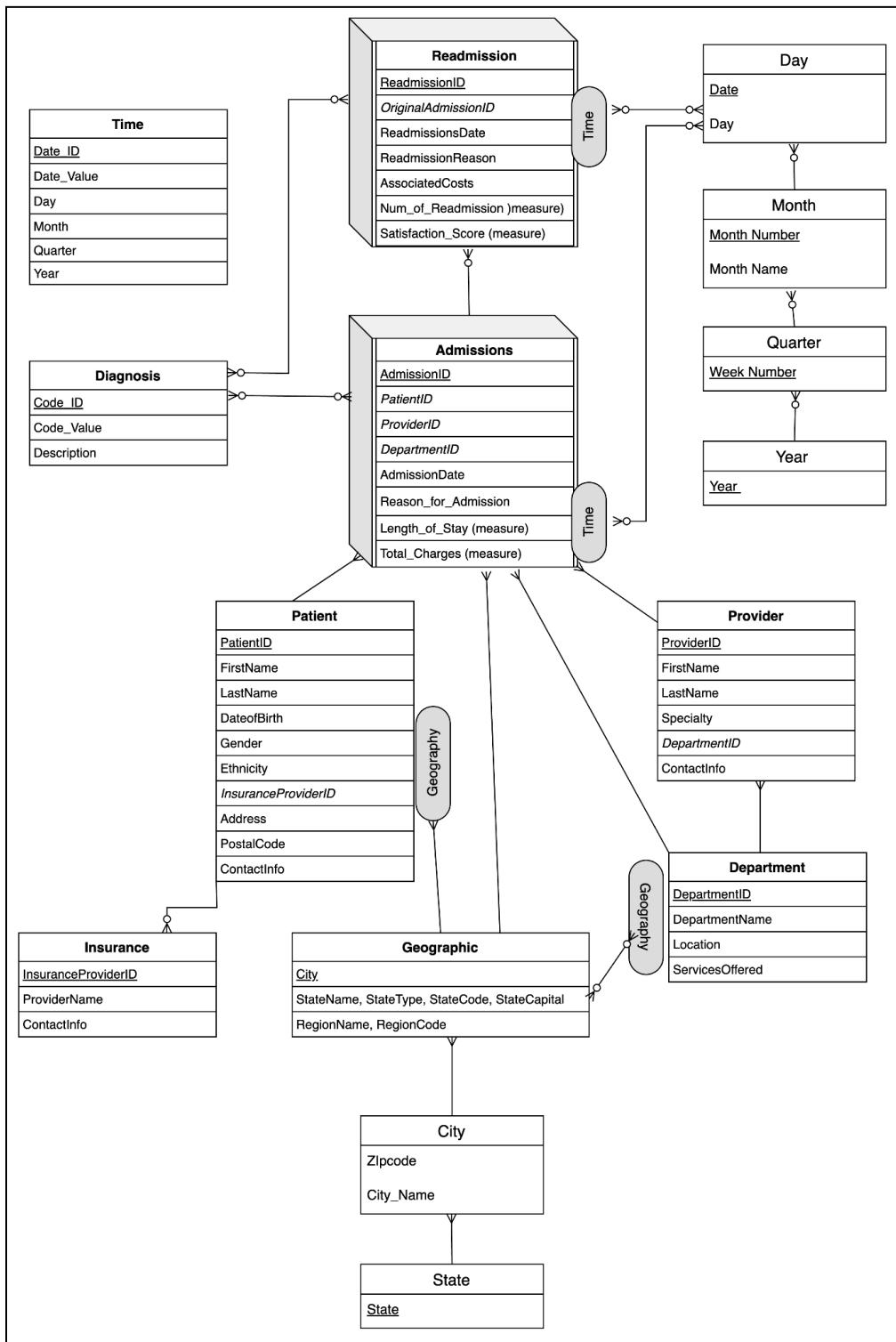
High-Level Business Processes

- Readmission Trend Analysis: Tracks readmission rates and identifies patterns in time and across departments.
- Cost and Revenue Analysis: Examines associated costs and charges by readmission events, insurance providers, and departments.
- Provider Performance: Assesses satisfaction scores and readmission rates by provider and department.
- Patient Demographic Analysis: Analyzes readmission trends across different patient demographics and geographic areas.
- Medication Efficacy: Monitors medication effectiveness by diagnosis and readmission rates.

Conceptual Data Warehouse (DW) Model

The Conceptual Data Warehouse Model for this project is as below:

Final Project 1 Report

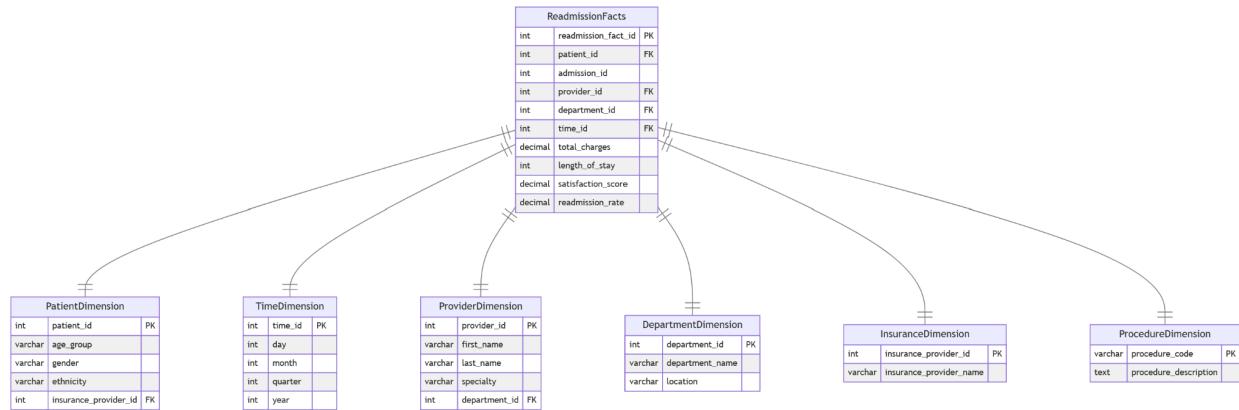


Final Project 1 Report

Logical Data Warehouse (DW) Model

This step is crucial for the PostgreSQL implementation and ensures that the relational model is logically sound and can support OLAP queries.

The logical data warehouse model maps the conceptual multidimensional model into a relational schema. This includes defining all tables, primary keys (PKs), foreign keys (FKs), references, and their relationships.



Fact Table: patient_readmissions:

```
CREATE TABLE patient_readmissions (
    readmission_id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES patient_dimension(patient_id),
    provider_id INTEGER REFERENCES provider_dimension(provider_id),
    admission_date INTEGER REFERENCES time_dimension(date_id),
    discharge_date INTEGER REFERENCES time_dimension(date_id),
    code_id INTEGER REFERENCES diagnosis_dimension(code_id),
    total_charges DECIMAL,--measure
    average_length_of_stay DECIMAL,--measure
    satisfaction_score INTEGER, --measure
    associated_costs DECIMAL -- measure
);
```

Dimension Tables:

- **Patient Dimension:**

```
CREATE TABLE patient_dimension (
    patient_id SERIAL PRIMARY KEY,
    age_group VARCHAR(20),
```

Final Project 1 Report

```
    gender VARCHAR(10),
    ethnicity VARCHAR(50),
    insurance_provider_id INTEGER
);
```

- **Time Dimension:**

```
CREATE TABLE time_dimension (
    date_id SERIAL PRIMARY KEY,
    day INTEGER,
    month INTEGER,
    quarter INTEGER,
    year INTEGER
);
```

- **Provider Dimension**

```
CREATE TABLE provider_dimension (
    provider_id SERIAL PRIMARY KEY,
    specialty VARCHAR(100),
    department_id INTEGER
);
```

- **Diagnosis Dimension:**

```
CREATE TABLE diagnosis_dimension (
    code_id SERIAL PRIMARY KEY,
    code_value VARCHAR(20),
    description TEXT
);
```

- **Geographic Dimension:**

```
CREATE TABLE geographic_dimension (
    location_id SERIAL PRIMARY KEY,
    city VARCHAR(50),
    state VARCHAR(50),
```

Final Project 1 Report

```
    region VARCHAR(50)  
);
```

This logical model maps directly from the conceptual model to relational tables in PostgreSQL, ensuring the correct structure for OLAP queries.

Database Implementation

The implementation of the relational schema for the hospital readmissions data warehouse was carried out in PostgreSQL. This schema adheres to the star schema design and includes a central fact table connected to multiple dimension tables through well-defined primary and foreign keys. The implementation aims to meet the "Exceeds Standard" criteria by enforcing data integrity using constraints and establishing the necessary relationships between tables to support efficient querying and analysis. Below is a detailed explanation of the steps taken to implement the database.

To ensure proper relational structure, each table was implemented with a primary key to uniquely identify each record. The fact table—which serves as the core for storing measurable events related to hospital readmissions—references all relevant dimension tables using foreign keys. This structure allows for efficient data analysis while maintaining referential integrity across the entire schema.

Fact Table: ReadmissionFacts

- The ReadmissionFacts table stores data related to each patient's readmission event. It includes foreign keys that reference various dimension tables, such as PatientDimension, TimeDimension, ProviderDimension, and others.
- The primary key for this table is readmission_fact_id, which uniquely identifies each record in the fact table. Foreign keys are used to connect to the relevant dimension tables.

Dimension Tables: Each dimension table contains a primary key that uniquely identifies records within the dimension. These keys are referenced by the fact table to maintain the integrity of the schema. For example, the PatientDimension table contains patient information and is linked to the fact table via the patient_id foreign key.

Constraints: To maintain data accuracy and ensure the integrity of the data within each table, several constraints were implemented:

- NOT NULL: This constraint ensures that fields essential to analysis, such as total_charges and length_of_stay, cannot be left empty. This guarantees that key measures are always available for analytical queries.
- CHECK: This constraint ensures that values meet specific criteria, such as the satisfaction score being within a valid range (1 to 5) and length of stay being a positive value.

Final Project 1 Report

- Referential Integrity: All foreign keys were defined with REFERENCES constraints to enforce the relationship between the fact table and the dimension tables. This prevents orphaned records in the fact table, ensuring that every fact has associated dimension data.

Implementation of Fact and Dimension Tables

In the PostgreSQL implementation, each fact and dimension table was created with proper SQL syntax to ensure accurate data modeling. Below is a brief overview of the key tables and their structures.

This Patients Readmissions table tracks all relevant events associated with hospital readmissions, such as the total charges, length of stay, satisfaction scores, and associated costs. The table references multiple dimension tables through foreign keys, ensuring that each readmission is contextualized with patient, provider, time, diagnosis, and geographic data.

Dimension tables such as PatientDimension, TimeDimension, ProviderDimension, DiagnosisDimension, and GeographicDimension provide the necessary descriptive context for analyzing the events stored in the fact table. These tables store attributes like patient demographics, time hierarchies, provider information, diagnostic codes, and geographic data.

Primary Events

The key business events tracked in the system are represented in the fact table. These events include Patient Readmission, Medication Administered, and Billing Records, each of which plays a major role in understanding and analyzing hospital readmissions.

- **Patient Readmission:** This is the primary event captured by the fact table. Each record tracks a specific readmission event for a patient, including details such as admission and discharge dates, length of stay, and associated costs. This event is central to the goal of analyzing and reducing hospital readmissions.
- **Medication Administered:** This event records the medications administered to patients during their readmission. By tracking this data, the system can help identify patterns between medication use and patient outcomes, informing future treatment decisions.
- **Billing Records:** Captures all financial data related to each readmission, including total charges, insurance claims, and patient out-of-pocket expenses. This data is essential for understanding the economic impact of readmissions on both the hospital and the patient.
- **Staff Scheduling and Provider Availability:** Tracks healthcare provider schedules, including shift details and provider availability during the patient's stay. Provider availability can impact patient outcomes, particularly in high-stress departments like Emergency. By correlating staff availability with readmission rates, the system can help identify potential staffing-related factors affecting patient care.

Final Project 1 Report

- **Medical Procedures:** Records all medical procedures performed during a patient's stay, including details like procedure type, date, and outcomes. By analyzing the procedures associated with readmitted patients, the system can help identify which procedures might correlate with higher or lower readmission rates, informing future care protocols.

Each of these events is directly connected to the business goals of reducing readmissions and optimizing patient care, making them important parts of the overall data warehouse model.

OLAP Operations

The OLAP operations outlined below can enable analysis in depth for healthcare administrators, facilitating data-driven decision-making to reduce readmissions and improve patient care. These operations leverage the multidimensional structure of the data warehouse to provide flexible, in-depth insights into key performance indicators.

- **Roll-Up**

Objective: Aggregate data to a higher level of summary to monitor trends at a broader time interval.

Example: ROLLUP*(Readmissions, AdmissionDate → Month, Department → Specialty, SUM(TotalCharges))

Description: This operation aggregates daily readmission costs to monthly totals, allowing administrators to observe spending patterns over a larger time frame. This overview aids in budget management by highlighting monthly or quarterly cost trends.

- **Drill-Down**

Objective: Access more granular details within specific departments or categories.

Example: DRILLDOWN(Readmissions (Department → Specialty))

Description: This drill-down operation enables administrators to break down hospital-wide readmission data by specialty, offering deeper insights into areas with higher readmission rates. By isolating data at the specialty level, administrators can target improvement efforts more effectively.

- **Slice**

Objective: Filter data to focus on a specific condition or demographic.

Example: SLICE(Readmissions, Diagnosis = 'Hypertension')

Description: Slicing allows administrators to isolate readmission data for a particular diagnosis, such as hypertension. This operation supports targeted resource allocation and interventions by focusing on conditions with potentially higher readmission risks.

- **Dice**

Objective: Filter data across multiple dimensions for a specific subgroup analysis.

Example: DICE(Readmissions, [AgeGroup = '40-50', InsuranceProvider = 'Provider A'])

Description: Dicing allows for multidimensional filtering. Here, it enables a focused analysis of patients aged 40-50 who are insured by Provider A, providing insights into readmission trends within this demographic and insurance category.

- **Pivot**

Objective: View data from alternative perspectives to identify demographic trends.

Final Project 1 Report

Example: PIVOT(Readmissions, (AgeGroup → X)

Description: This operation enables administrators to pivot the data, comparing readmission rates or costs across age groups and gender. Pivoting reveals demographic patterns, assisting in identifying trends in healthcare needs among different patient segments.

- **Drill-Across**

Objective: Compare related data across different specialties or departments.

Example: DRILLACROSS(Readmissions, Specialty, [Provider])

Description: The drill-across operation provides comparative insights across providers within specialties. By analyzing readmission rates across these dimensions, administrators can identify differences in provider performance and determine where additional resources or support may be necessary.

- **Rename**

Objective: To change the name following business requirements.

Example: RENAME(HospitalReadmissionsCube,

(PatientDimension.AgeGroup --> PatientAgeGroup),

(ProviderDimension.Specialty --> ProviderSpecialization),

(DepartmentDimension.DepartmentName --> Department)

);

Description: The rename operation in OLAP allows administrators to change the names based on the business needs.

By applying roll-up, drill-down, slice, dice, pivot, drill-across and rename, administrators gain insights that go beyond basic numbers. These operations reveal patterns, enable forecasting, and highlight areas for improvement, ultimately helping to make decisions that benefit both patients and the healthcare system as a whole.

Final Project 1 Report

Milestone 4

Project Title: Optimizing Patient Care: A Hospital Readmissions Database

Details from Previous Milestones:

Milestone 1: Project Initiation and Problem Definition

The project titled "Optimizing Patient Care: A Hospital Readmissions Database" aims to address the high cost and operational impact of hospital readmissions. In this milestone, we defined the problem and identified key objectives such as reducing readmission rates, improving patient outcomes, and ensuring data privacy and security (HIPAA compliance). A Kaggle dataset was selected for analysis, and PostgreSQL was chosen to implement the operational database as part of the course requirement. The design includes tracking patient demographics, medical history, treatment data, and key performance indicators to support predictive analytics and efficient data management.

Milestone 2: Data Modeling and Schema Design

This milestone covered designing an Extended Entity-Relationship Diagram (ERD) and mapping it to a normalized relational model. Here, key data entities include patients, healthcare providers, admissions, medical procedures, and readmission records, among others. We also identified potential dimensions and measures for analytics, such as the patient, time, provider, and diagnosis dimensions. The model was implemented in PostgreSQL, establishing primary and foreign key constraints to ensure data integrity and support efficient OLAP operations.

Milestone 3: Multidimensional Data Model and Data Warehouse Implementation

In this milestone, we created a star schema-based data warehouse to support OLAP queries. We linked a core fact table for readmission metrics to several dimensions, including patient, provider, department, time, and procedure. We implemented OLAP operations such as slicing, dicing, roll-up, and drill-down to enable detailed analysis across patient demographics, treatment types, and healthcare provider performance. This setup allows healthcare administrators to identify trends, optimize patient care, and reduce readmissions through data-driven decision-making.

Stakeholder Impact

With fully operational tables, optimized ETL flows, and secure OLAP functionalities, this milestone marks a pivotal step in delivering actionable insights on hospital readmissions. This data warehouse provides a powerful tool for healthcare administrators, enabling data-driven decisions to reduce readmission rates and improve patient outcomes.

Final Project 1 Report

ETL Execution

1. Sourcing Data from the Operational Database

The primary data source for the data warehouse is the operational database designed in previous milestones. This database captures a wide range of hospital-related data, including patient admissions, readmissions, discharge details, medical procedures, billing records, and patient feedback. The operational database is updated with new transactions, and this data flows into the data warehouse. We used 2 data sources here and also synthesized data as well.

2. Data Warehouse Table Loading and Surrogate Key Implementation

We utilized the schema and detailed entity relationships established in Milestone 2, where core entities like patient demographics, departments, providers, and admissions were defined. The schema, sourced from a CSV file that updates patient dimensions, Kaggle and the Centers for Medicare & Medicaid Services as identified in Milestone 1, is now fully populated. Additional data was also artificially generated using Python.

3. Insert and Update Flows

Our ETL processes are configured to handle both inserts and updates, ensuring continuous data integrity and completeness for dynamic entities such as patient admissions and provider information. These flows were designed to meet the quality standards discussed in Milestone 1.

4. Dimensional and Fact Table Population with ETL Prepopulations

Leveraging the star schema structure from Milestone 3, the Readmission Facts table serves as the fact table, linking to prepopulated dimensions (Patient, Provider, Diagnosis, etc.) for efficient OLAP queries. This organization provides a streamlined data flow and supports optimal analysis.

5. Transformation Types and Additional Features

Our transformation approach is based on data volatility requirements identified in Milestone 2. For instance, patient demographics are managed through Type 1 transformations, while changes in insurance provider information are recorded as Type 2 Slowly Changing Dimensions (SCDs), preserving historical context.

6. Advanced Analytics, OLAP Operations, Future Analysis

Expanding on Milestone 3's star schema, the OLAP-enabled database supports advanced analytical capabilities. We incorporated roll-up operations (e.g., monthly to quarterly aggregations), drill-downs by department and provider specialty, and pivot views for cross-sectional comparisons across patient demographics.

1. Readmission Trends Analysis

- Objective: Identify patterns in patient readmissions to determine which demographics, departments, and diagnoses have the highest readmission rates.

Final Project 1 Report

- Our Approach:
 - Aggregate readmissions by patient demographics (age, ethnicity, gender) and department.
 - Use time-series analysis to observe trends in readmission rates by diagnosis and department.
- Possible Insights: Highlight departments or demographic groups with higher-than-average readmission rates, supporting targeted interventions or resource allocation for these areas.

2. Cost Analysis by Diagnosis and Department

- Objective: Understand which diagnoses and departments incur the highest costs, supporting budget optimization and resource allocation.
- Our Approach:
 - Use the Avg_cost_by_diagnosis aggregation to identify high-cost conditions.
 - Break down total charges by department and diagnosis to compare against patient outcomes.
- Possible Insights: Pinpoint costly diagnoses and departments that could benefit from efficiency improvements or cost-saving initiatives.

3. Patient Satisfaction Analysis

- Objective: Assess patient satisfaction scores by provider and department to identify strengths and areas for improvement in patient experience.
- Our Approach:
 - a. Analyze the Avg_satisfaction_by_provider metric to assess each provider's performance.
 - b. Compare satisfaction scores by department and patient demographics to identify any trends or disparities.
- Possible Insights: Identify high-performing providers and departments in terms of patient satisfaction, which can help in replicating best practices across the hospital.

4. Insurance Impact on Readmissions and Costs

- Objective: Evaluate how different insurance providers affect readmission rates and treatment costs, which may indicate disparities in access to care or coverage quality.
- Our Approach:
 - a. Track readmissions and treatment costs by insurance provider using SCD Type 2 historical data.
 - b. Compare readmission rates and costs for different insurance plans over time.
- Possible Insights: Reveal patterns that may suggest how insurance type impacts patient outcomes and cost structures, potentially guiding hospital negotiations with insurers or highlighting areas for patient support.

5. Provider Performance Analysis by Department

- Objective: Evaluate provider performance across different departments based on patient outcomes and satisfaction.
- Our Approach:
 - Compare readmission rates and satisfaction scores by provider, broken down by department.

Final Project 1 Report

- Identify any providers with consistently high or low performance across multiple metrics.
- Possible Insights: Help identify high-performing providers who may be mentors or leaders within their departments, as well as those who may benefit from additional support.

6. Forecasting Readmission Rates

- Objective: Use historical data to forecast future readmission rates, enabling proactive planning for staffing and resource allocation.
- Our Approach:
 - Apply time-series forecasting techniques on readmission rates across departments and diagnoses.
 - Identify seasonal trends or fluctuations that could inform resource planning and preventive measures.
- Possible Insights: Anticipate potential surges in readmissions, allowing the hospital to adjust resources, improve discharge planning, or implement preventive interventions.

7. Correlation Analysis between Demographics and Outcomes

- Objective: Discover correlations between patient demographics (age, gender, ethnicity) and healthcare outcomes, such as readmission rates and satisfaction.
- Our Approach:
 - Use correlation analysis to assess relationships between demographics and outcomes.
 - Segment data by age group, ethnicity, or gender to understand variations in patient experiences or outcomes.
- Possible Insights: Gain an understanding of demographic factors impacting patient outcomes, guiding targeted interventions for specific groups.

7. Transformations

Transformations play a vital role in preparing the data for analytical queries. Two main types of transformations are applied:

- **Type 1 Transformations (Overwrites):** These transformations update non-historical attributes, such as patient contact information, directly overwriting old values with new ones.
- **Type 2 Transformations (Slowly Changing Dimensions):** These transformations track historical changes by creating new records for attributes that require versioning, such as changes in a patient's insurance provider. This approach preserves historical data, which is critical for trend analysis and reporting.

Please refer to the Appendix (ii. Talend Implementation, screenshot #3) to see further details.

8. Data Aggregations

Data aggregations are implemented to support high-level reporting and OLAP operations.

1. **Average Cost by Diagnosis (Avg_cost_by_diagnosis):** This aggregation calculates the average cost associated with each diagnosis in the dataset. This measure is essential for identifying the financial impact of various medical conditions on hospital

Final Project 1 Report

resources. By examining costs across diagnoses, healthcare administrators can pinpoint which conditions are more resource-intensive. This insight can guide strategic planning, such as optimizing treatment protocols for high-cost diagnoses or allocating resources more effectively.

Calculation:

- *Input:* Records of patient admissions, including fields like DiagnosisCode and Cost
- *Output:* A summary table that shows each DiagnosisCode alongside an Avg_Cost, representing the mean cost associated with that particular diagnosis.

The screenshot shows a database management interface with two main panes. On the left, the Object Explorer displays a tree view of database objects: Sequences, Tables (31), Constraints, Indexes, RLS Policies, Rules, Triggers, and various cleaned and dimension tables like admissions_cleaned, average_cost_by_diagnosis, avg_satisfaction_by_provider, and geographic_dimension. The 'average_cost_by_diagnosis' table is currently selected, showing its four columns: code_id, code_value, description, and avg_cost_per_diagnosis. On the right, a query results window titled 'Project 1 - Healthcare Database/postgres@IE 6750 Project 1' shows the results of the query 'select * from average_cost_by_diagnosis'. The results are presented in a table with columns: code_id, code_value, description, and avg_cost_per_diagnosis. The data includes rows for various medical codes and their corresponding average costs.

code_id	code_value	description	avg_cost_per_diagnosis
1	229	Injury	12487.27
2	227	Musculoskeletal	12480.60
3	226	Digestive	12482.37
4	225	Missing	13610.44
5	223	Diabetes	12524.30
6	224	Other	12522.49
7	222	Circulatory	12463.71
8	228	Respiratory	12500.99

2. Average Satisfaction by Provider (Avg_satisfaction_by_provider): This aggregation provides the average patient satisfaction score for each healthcare provider. This metric offers a valuable perspective on provider performance from the patient's viewpoint. Consistently high satisfaction scores often reflect positive patient experiences, while lower scores can highlight areas where patient interactions, treatment approaches, or communication might need improvement.

Calculation:

- *Input:* Patient satisfaction data with fields like ProviderID and SatisfactionScore
- *Output:* A table that aggregates each ProviderID with an Avg_Satisfaction, showing the average satisfaction score attributed to each provider.

Final Project 1 Report

The screenshot shows a PostgreSQL database interface. On the left, the Object Explorer displays a tree view of database objects, including Sequences, Tables (31), Columns, Constraints, Indexes, RLS Policies, Rules, Triggers, and various tables like admissions, admissions_cleaned, average_cost_by_diagnosis, avg_satisfaction_by_provider, and geographic_dimension. The 'avg_satisfaction_by_provider' table is currently selected. On the right, the main area shows a query editor with the SQL command: 'select * from avg_satisfaction_by_provider'. Below the query is a table titled 'Data Output' with columns: avg_satisfaction_score, provider_id, and specialty. The table contains 19 rows of data, each with a provider ID of 3 and a specialty of 'Cardiology'. The total number of rows is 69, and the query was completed in 0.00:00.150.

avg_satisfaction_score	provider_id	specialty
1	3	Cardiology
2	3	Cardiology
3	3	Cardiology
4	3	Cardiology
5	3	Cardiology
6	3	Cardiology
7	3	Cardiology
8	3	Cardiology
9	3	Cardiology
10	3	Cardiology
11	3	Cardiology
12	3	Cardiology
13	3	Cardiology
14	3	Cardiology
15	3	Cardiology
16	3	Cardiology
17	3	Neurology
18	3	Cardiology
19	3	Cardiology

9. Complete and Incremental Loads

To maintain an up-to-date data warehouse, we used two types of loading strategies:

- **Complete Loads:** A full load is performed during the initial setup of the hospital readmissions data warehouse or in rare cases when a total reload is necessary, such as after a major schema modification or when adding a new data source. This process ensures that all data is freshly loaded, providing a clean, comprehensive baseline for subsequent analysis and reporting.

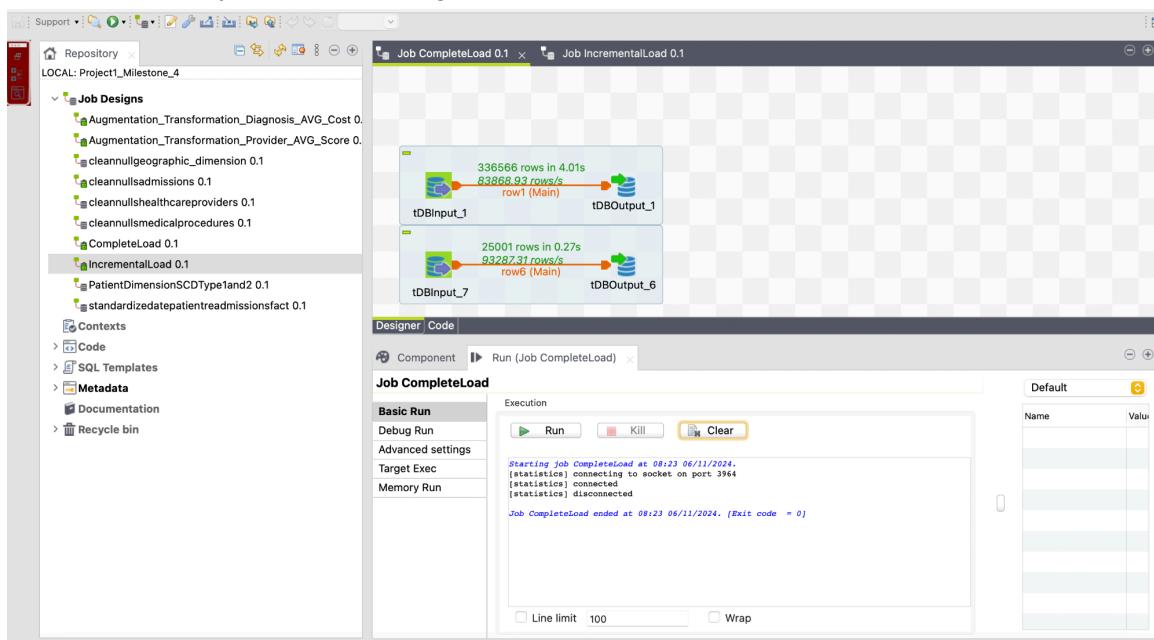


Image: Complete Load Example in Talend.

Final Project 1 Report

- **Incremental Loads:** For ongoing updates, incremental loads capture only new or modified records since the previous load, using time-stamped changes in the hospital readmissions database. This ensures the data warehouse stays current with minimal data transfer and processing overhead.

Changes to patient details or insurance information are updated incrementally, capturing only the modifications since the last ETL run.

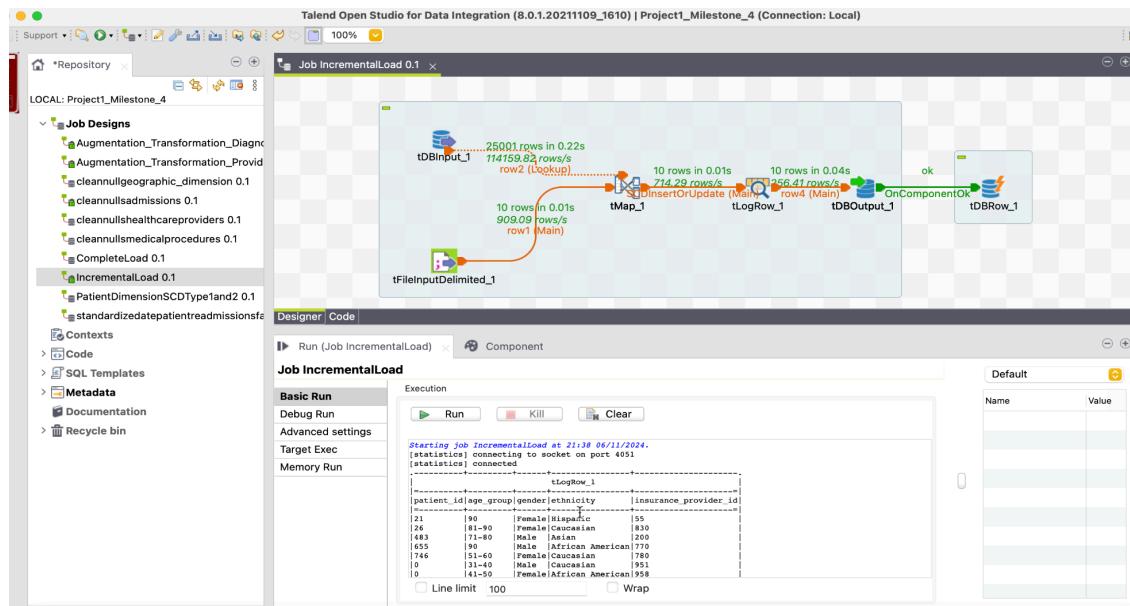


Image: Incremental load example in Talend.

The screenshot shows the pgAdmin interface connected to a PostgreSQL database. The left sidebar shows the schema structure under 'data_warehouse'. A query is run against the 'control_table' in the 'data_warehouse' schema:

```
1 Select * from data_Warehouse.control_table
```

The results are displayed in a table:

jobname	lastloaddate	rowcount
IncrementalLoad	2024-11-06 21:38:47.38597	10

At the bottom, the status bar indicates: 'Total rows: 1 of 1 Query complete 00:00:00.139 Ln 1, Col 43'.

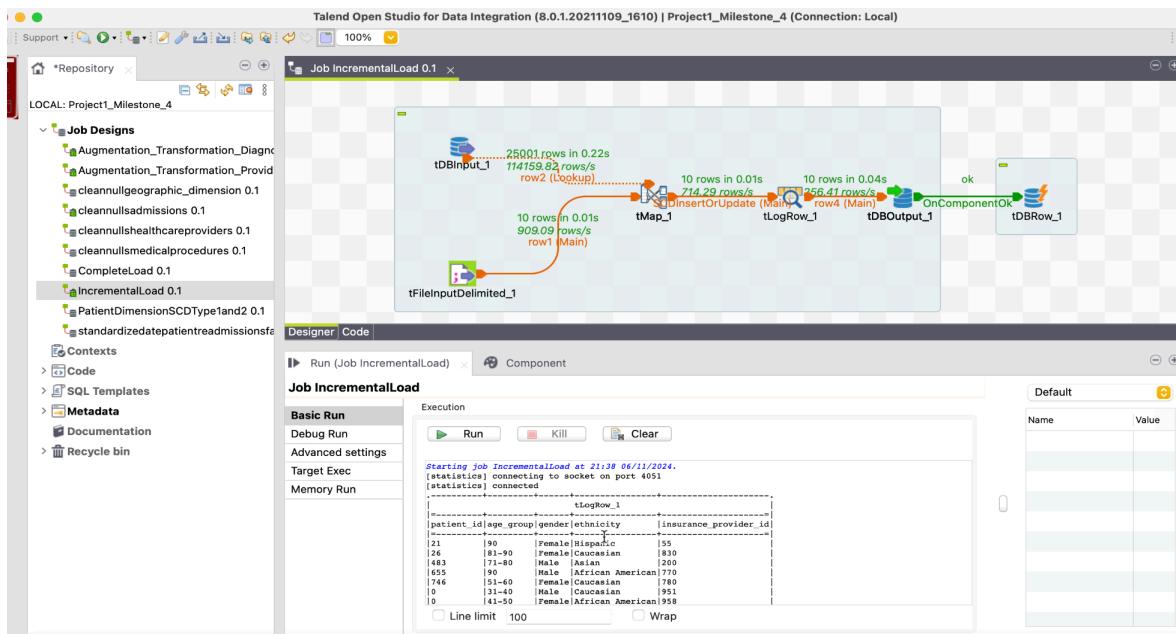
Final Project 1 Report

10. Control Flow Documentation

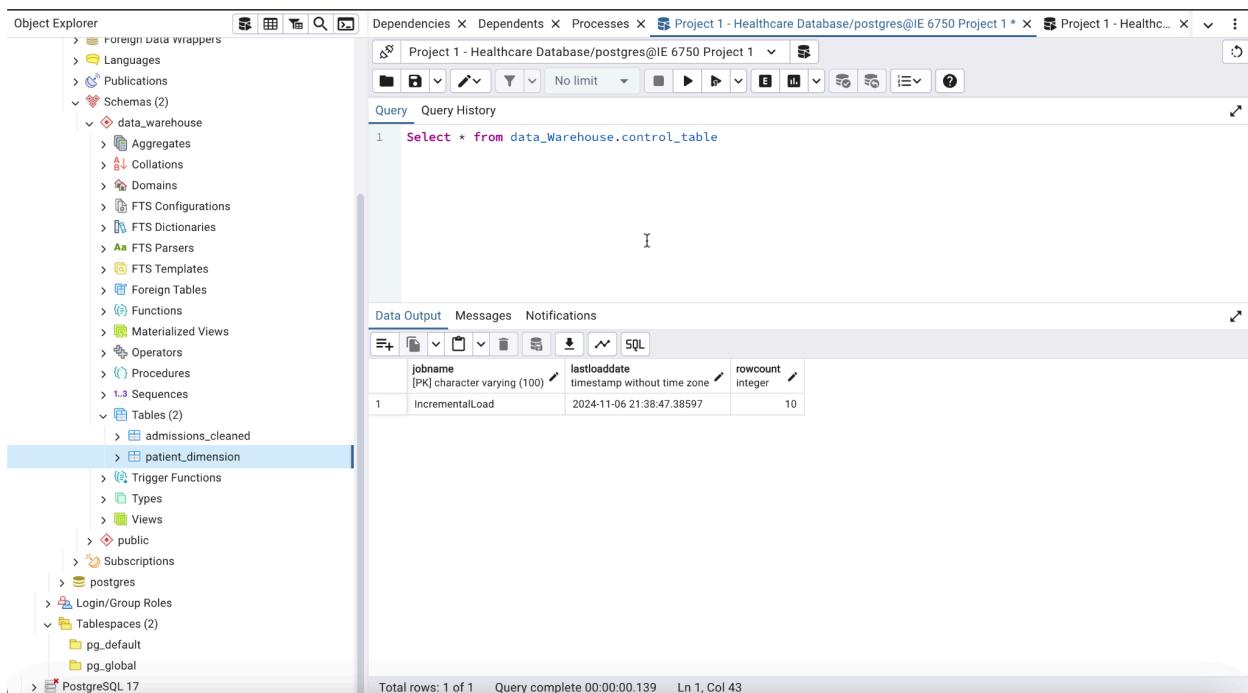
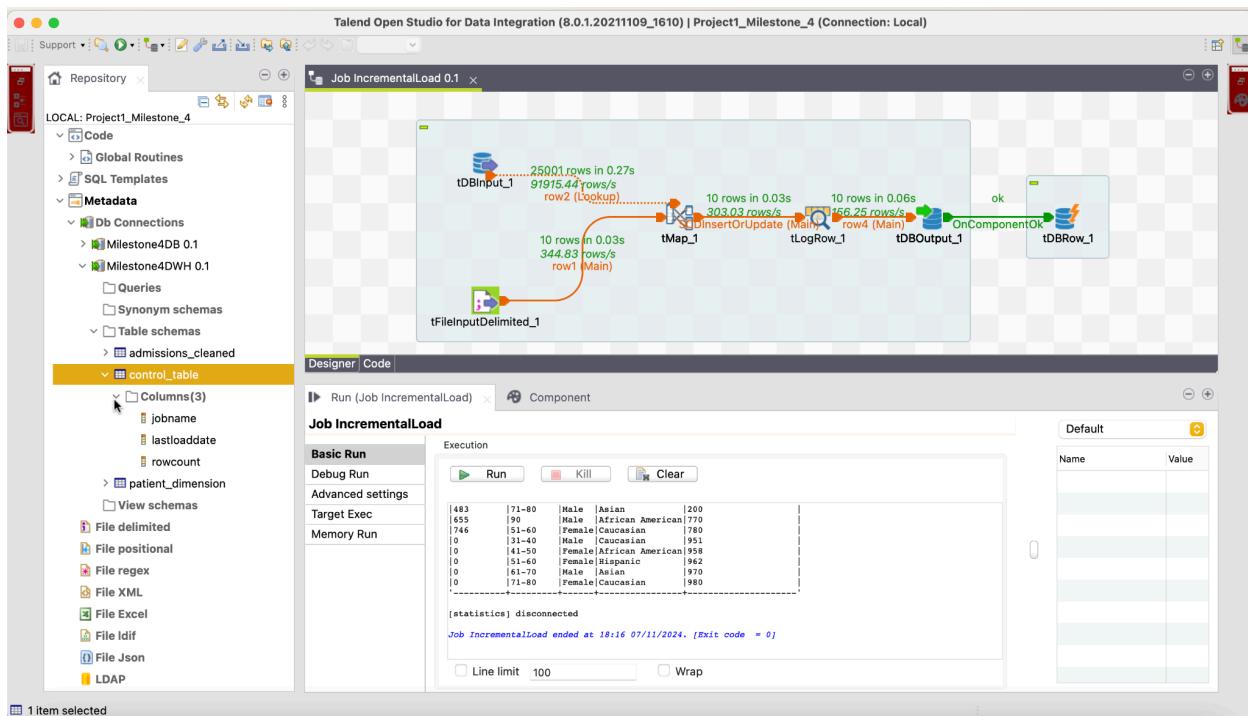
A dedicated control table documents each ETL execution, capturing metadata, ensuring accountability and enabling quick diagnostics. Managed by the wrapper job, this control table logs each ETL job in sequence, capturing essential details like job name, load date/time, and row count. The wrapper job orchestrates the ETL workflow, ensuring that jobs are executed in the correct order and updating the control table with metadata for each job as it completes. This setup provides a centralized record, offering visibility into the ETL workflow and enabling quick troubleshooting.

By documenting when and how many rows are processed for each job, the control table ensures incremental loads occur as expected and that the data warehouse remains up-to-date. This table is invaluable for monitoring and auditing, allowing the data engineering team to verify job execution, maintain data integrity across the pipeline, and support both data governance and compliance efforts.

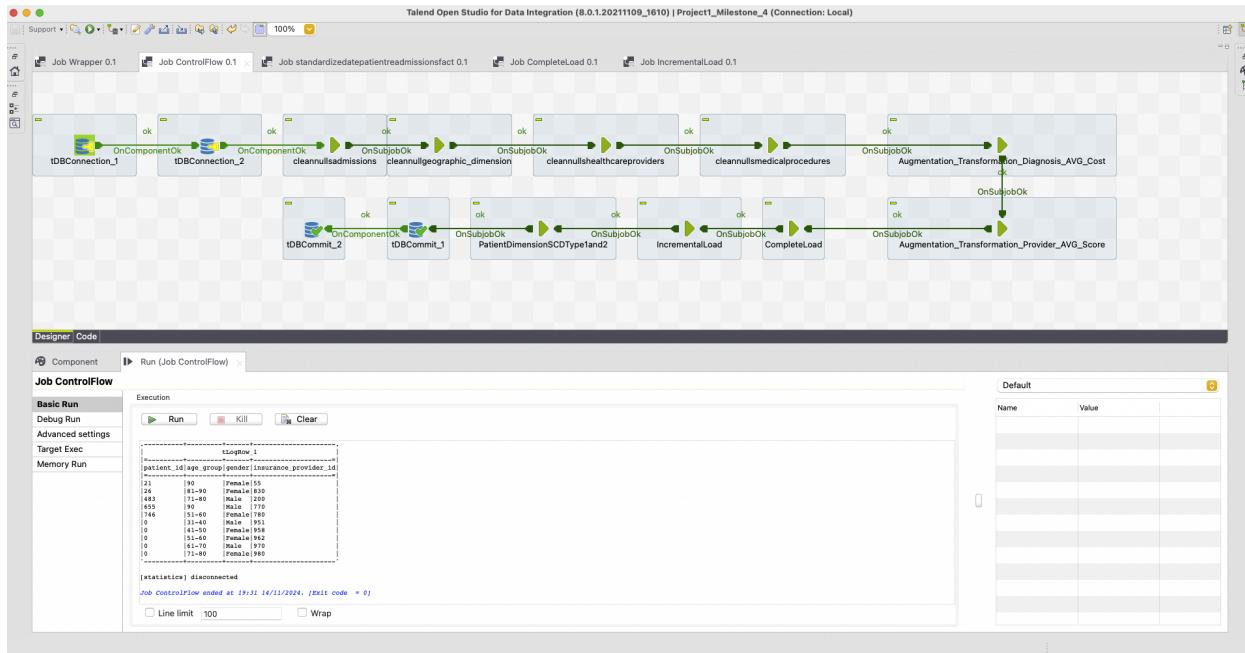
In the future, this control table will remain an essential resource for operational monitoring and auditing. With a clear log of each ETL job and the sequence managed by the wrapper job, the data engineering team can quickly verify if jobs are running as scheduled and identify any points of failure. For instance, if a load fails or an error is recorded, the team can rely on this table to assess and address the issue without manual guesswork. Additionally, having a timestamped log of all ETL activities managed through the wrapper job supports data governance and compliance efforts, ensuring that data integrity is maintained consistently throughout the pipeline.



Final Project 1 Report



Final Project 1 Report



Conclusion

In this project, we developed a comprehensive data warehouse for tracking hospital readmissions, using PostgreSQL for data storage and Talend for designing and managing the ETL workflows. From initial data sourcing to implementing both complete and incremental loads, we worked to ensure that our data warehouse remains accurate, current, and valuable for analyzing readmission trends.

Throughout the project, we encountered several challenges. One of the key issues was configuring incremental loads accurately to capture only new or modified records without duplicating data. We addressed this by carefully implementing time-stamped filters in Talend, allowing us to update our warehouse efficiently without unnecessary data processing. Another challenge was ensuring accurate row counts and tracking in our control table to maintain transparency across ETL jobs. By adding detailed logging and error-handling steps, we were able to monitor job status effectively and quickly troubleshoot any inconsistencies.

Using Talend, we configured ETL workflows with necessary transformations, aggregations, and control tracking, logging each job's details—such as load timestamps and row counts—to ensure accountability and quick diagnostics. Aggregated metrics, including average costs by diagnosis and satisfaction levels by provider, offer healthcare administrators actionable insights into factors impacting readmissions and resource allocation.

We aimed to demonstrate the potential of a well-structured data warehouse to support data-driven decision-making in healthcare. Future enhancements could involve integrating additional data sources, optimizing ETL performance, and exploring predictive analytics to

Final Project 1 Report

further support patient care and operational efficiency. This experience provided us with valuable hands-on knowledge in data warehousing, ETL workflows, and healthcare analytics, skills that will be instrumental in our professional and academic development.

Appendix:

i. Postgres Implementation

- **Admissions**

The screenshot shows the pgAdmin interface with the 'Object Explorer' on the left and a 'Query Results' window on the right. The Object Explorer displays a tree structure of database objects, including tables like 'admissions', 'admissions_cleaned', and 'average_cost_by_diagnosis'. The 'admissions' table is currently selected. The Query Results window shows the output of the SQL query: 'select * from admissions_cleaned'. The results grid contains 19 rows of data, each representing an admission record with columns such as admission_id, patient_id, admission_date, reason_for_admission, department_id, provider_id, and begin_date.

admission_id	patient_id	admission_date	reason_for_admission	department_id	provider_id	begin
740656	15656	2020-04-29	Description for Digestive	1	1	
50492	493	2020-12-02	Unknown	98	905	
50561	501	2020-03-08	Unknown	98	905	
50517	517	2020-07-05	Unknown	98	905	
50543	543	2020-12-28	Unknown	98	910	
50550	550	2020-11-19	Unknown	98	905	
50623	623	2020-09-04	Unknown	98	910	
50654	654	2020-12-27	Unknown	98	910	
50680	680	2020-06-05	Unknown	98	910	
50692	692	2020-04-30	Unknown	98	905	
50700	700	2020-02-15	Unknown	98	905	
50705	705	2020-06-16	Unknown	98	905	
50720	720	2020-02-04	Unknown	98	905	
50725	725	2020-05-09	Unknown	98	905	
50739	739	2020-03-26	Unknown	98	910	
50742	742	2020-05-07	Unknown	98	905	
50754	754	2020-09-07	Unknown	98	910	
50766	766	2020-10-29	Unknown	98	905	
50770	770	2020-12-14	Unknown	98	910	

- **Avg_cost_by_diagnosis (Aggregation performed by Talend)**

The screenshot shows the pgAdmin interface with the 'Object Explorer' on the left and a 'Query Results' window on the right. The Object Explorer displays a tree structure of database objects, including tables like 'average_cost_by_diagnosis' and 'code'. The 'average_cost_by_diagnosis' table is currently selected. The Query Results window shows the output of the SQL query: 'select * from average_cost_by_diagnosis'. The results grid contains 8 rows of data, each representing a diagnosis code with columns such as code_id, code_value, description, and avg_cost_per_diagnosis.

code_id	code_value	description	avg_cost_per_diagnosis
229	Injury	Description for Injury	12487.27
227	Musculoskeletal	Description for Musculoskeletal	12480.60
226	Digestive	Description for Digestive	12482.37
225	Missing	Description for Missing	13610.44
223	Diabetes	Description for Diabetes	12524.30
224	Other	Description for Other	12522.49
222	Circulatory	Description for Circulatory	12463.71
228	Respiratory	Description for Respiratory	12500.99

Final Project 1 Report

- Avg. satisfaction by provider (Aggregation done by Talend)

The screenshot shows a database interface with the following details:

- Object Explorer:** On the left, it lists various database objects including tables, sequences, constraints, indexes, RLS policies, rules, and triggers.
- Query Editor:** In the center, a query is being run: `select * from avg_satisfaction_by_provider`.
- Data Output:** On the right, the results of the query are displayed in a table. The table has three columns: `avg_satisfaction_score` (integer), `provider_id` (PK integer), and `specialty` (character varying (100)). The data shows 19 rows of results, all with a satisfaction score of 3, provider IDs ranging from 1363 to 1565, and specialty as Cardiology.

	avg_satisfaction_score	provider_id	specialty
1	3	1363	Cardiology
2	3	1330	Cardiology
3	3	1586	Cardiology
4	3	1425	Cardiology
5	3	1553	Cardiology
6	3	1264	Cardiology
7	3	1392	Cardiology
8	3	1454	Cardiology
9	3	1404	Cardiology
10	3	1627	Cardiology
11	3	1338	Cardiology
12	3	1561	Cardiology
13	3	1272	Cardiology
14	3	1623	Cardiology
15	3	1334	Cardiology
16	3	1301	Cardiology
17	3	1	Neurology
18	3	1631	Cardiology
19	3	1565	Cardiology

Total rows: 69 of 69 Query complete 00:00:00.150 Ln 1, Col 43

- Billing Records

The screenshot shows a database interface with the following details:

- Object Explorer:** On the left, it lists various database objects including tables, sequences, constraints, indexes, RLS policies, rules, and triggers.
- Query Editor:** In the center, a query is being run: `select * from billingrecords`.
- Data Output:** On the right, the results of the query are displayed in a table. The table has seven columns: `billing_id` (PK integer), `admission_id` (integer), `total_charge` (numeric (12,2)), `insurance_provider_id` (integer), `claim_status` (character varying (50)), and `patient_out_of_pocket` (numeric (12,2)). The data shows 19 rows of results, all with a total charge between 725001 and 725019, insurance provider ID 1, claim status Pending or Approved, and patient out-of-pocket amounts ranging from 2825.00 to 1777.00.

	billing_id	admission_id	total_charge	insurance_provider_id	claim_status	patient_out_of_pocket
1	1	725001	11756.00	1	Pending	2825.00
2	2	725002	8182.00	1	Approved	2482.00
3	3	725003	15093.00	1	Denied	2129.00
4	4	725004	10040.00	1	Approved	2761.00
5	5	725005	13210.00	1	Approved	2723.00
6	6	725006	8862.00	1	Denied	1123.00
7	7	725007	13796.00	1	Pending	1217.00
8	8	725008	7054.00	1	Pending	2363.00
9	9	725009	6462.00	1	Pending	859.00
10	10	725010	9635.00	1	Pending	1488.00
11	11	725011	8408.00	1	Pending	2263.00
12	12	725012	13566.00	1	Pending	2332.00
13	13	725013	13920.00	1	Denied	647.00
14	14	725014	8011.00	1	Approved	2047.00
15	15	725015	19581.00	1	Pending	2536.00
16	16	725016	8525.00	1	Denied	746.00
17	17	725017	9766.00	1	Denied	1151.00
18	18	725018	9299.00	1	Approved	899.00
19	19	725019	16602.00	1	Pending	1777.00

Total rows: 1000 of 250000 Query complete 00:00:00.342 Ln 1, Col 29

Final Project 1 Report

- Diagnosis dimension

Object Explorer

```

    > 1.3 Sequences
    > Tables (31)
        > admissions
        > admissions_cleaned
        > average_cost_by_diagnosis
        > avg_satisfaction_by_provider
        > billingrecords
        > diagnosis_dimension
            > Columns
            > Constraints
            > Indexes
            > RLS Policies
            > Rules
            > Triggers
        > discharges
        > equipment
        > geographic_dimension
        > geographic_dimension_cleaned
        > healthcareproviders
        > healthcareproviders_cleaned
        > hospitaldepartments
        > insurance_dimensions
        > insurance_providers
        > laboratorytests
        > medicalcodes
        > medicalprocedures
        > medicalprocedures_cleaned
        > medications
        > medicationsadministered
        > patient_dimension
        > patient_readmissions_fact
        > patient_readmissions_fact_cleaned
        > patientfeedback
        > patients
        > procedure_dimension
        > provider_dimension
        > readmissionrecords
        > staffscheduling
        > time_dimension
        > time_dimension
        > Trigger Functions
        > Types

```

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 X

Query Query History

```
1 select * from diagnosis_dimension
```

Data Output Messages Notifications

	code_id [PK] integer	code_value character varying(20)	description text
1	1	E11.9	Type 2 diabetes mellitus without complications
2	2	I10	Essential (primary) hypertension
3	3	0W3POZZ	Insertion of Infusion Device into Subcutaneous Tissue
4	4	3EDAA3MZ	Introduction of Other Antineoplastic into Peripheral Ve...
5	5	E11.9	Type 2 diabetes mellitus without complications
6	6	I10	Essential (primary) hypertension
7	7	0W3POZZ	Insertion of Infusion Device into Subcutaneous Tissue
8	8	3EDAA3MZ	Introduction of Other Antineoplastic into Peripheral Ve...
9	9	E11.9	Type 2 diabetes mellitus without complications
10	10	I10	Essential (primary) hypertension
11	11	0W3POZZ	Insertion of Infusion Device into Subcutaneous Tissue
12	12	3EDAA3MZ	Introduction of Other Antineoplastic into Peripheral Ve...
13	13	E11.9	Type 2 diabetes mellitus without complications
14	14	I10	Essential (primary) hypertension
15	15	0W3POZZ	Insertion of Infusion Device into Subcutaneous Tissue
16	16	3EDAA3MZ	Introduction of Other Antineoplastic into Peripheral Ve...
17	17	E11.9	Type 2 diabetes mellitus without complications
18	18	I10	Essential (primary) hypertension
19	19	0W3POZZ	Insertion of Infusion Device into Subcutaneous Tissue

Total rows: 324 of 324 Query complete 00:00:00 175 Ln 1 Col 1

- Discharges

Object Explorer

```

    > admissions
        > Columns
        > Constraints
        > Indexes
        > RLS Policies
        > Rules
        > Triggers
    > admissions_cleaned
    > average_cost_by_diagnosis
        > Columns (4)
            > code_id
            > code_value
            > description
            > avg_cost_per_diagnosis
        > Constraints
        > Indexes
        > RLS Policies
        > Rules
        > Triggers
    > avg_satisfaction_by_provider
        > Columns
        > Constraints
        > Indexes
        > RLS Policies
        > Rules
        > Triggers
    > billingrecords
    > diagnosis_dimension
        > discharges
            > Columns
            > Constraints
            > Indexes
            > RLS Policies
            > Rules
            > Triggers
        > equipment
    > geographic_dimension
        > Columns (3)
            > location_id
            > city
            > state
        > Constraints

```

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 X

Query Query History

```
1 select * from discharges
```

Data Output Messages Notifications

	discharge_id [PK] integer	patient_id integer	discharge_date date	discharge_disposition character varying(100)	follow_up_instructions text
1	1	1	2020-10-25	Transferred to another hospital	Follow up with primary care physician in 2 weeks.
2	2	2	2020-04-28	Transferred to another hospital	Follow up with primary care physician in 2 weeks.
3	3	3	2020-08-16	Discharged to home	Follow up with primary care physician in 2 weeks.
4	4	4	2020-12-30	Discharged to home	Follow up with primary care physician in 2 weeks.
5	5	5	2020-10-07	Transferred to another hospital	Follow up with primary care physician in 2 weeks.
6	6	6	2020-11-26	Discharged to home	Follow up with primary care physician in 2 weeks.
7	7	7	2020-12-17	Discharged to home	Follow up with primary care physician in 2 weeks.
8	8	8	2020-05-06	Left against medical advice	Follow up with primary care physician in 2 weeks.
9	9	9	2020-08-15	Transferred to another hospital	Follow up with primary care physician in 2 weeks.
10	10	10	2020-04-25	Discharged to home	Follow up with primary care physician in 2 weeks.
11	11	11	2020-09-17	Left against medical advice	Follow up with primary care physician in 2 weeks.
12	12	12	2020-04-02	Discharged to home	Follow up with primary care physician in 2 weeks.
13	13	13	2020-06-21	Left against medical advice	Follow up with primary care physician in 2 weeks.
14	14	14	2020-12-20	Discharged to home	Follow up with primary care physician in 2 weeks.
15	15	15	2020-05-16	Transferred to another hospital	Follow up with primary care physician in 2 weeks.
16	16	16	2020-11-29	Discharged to home	Follow up with primary care physician in 2 weeks.
17	17	17	2020-02-08	Left against medical advice	Follow up with primary care physician in 2 weeks.
18	18	18	2020-05-24	Discharged to home	Follow up with primary care physician in 2 weeks.
19	19	19	2020-05-20	Transferred to another hospital	Follow up with primary care physician in 2 weeks.

Total rows: 1000 of 25001 Query complete 00:00:00 108 Ln 2, Col 1

Final Project 1 Report

- Equipment

The screenshot shows a PostgreSQL database interface with the following details:

- Object Explorer:** Shows a tree view of tables, including 'admissions', 'admissions_cleaned', 'average_cost_by_diagnosis', 'avg_satisfaction_by_provider', 'billingrecords', 'diagnosis_dimension', 'discharges', 'equipment' (selected), 'etl_job_control', 'fact_duplicate', 'geographic_dimension', 'geographic_dimension_cleaned', 'healthcareproviders', 'healthcareproviders_cleaned', 'hospitaldepartments', 'insurance_dimension', 'insuranceproviders', 'laboratorytests', 'medicalcodes', 'medicalprocedures', 'medicalprocedures_cleaned', 'medications', 'medicationsadministered', 'patient_dimension', 'patient_readmissions_fact', 'patient_readmissions_fact_cleaned', 'patientfeedback', 'patients', 'provider_dimension', 'readmissionrecords', 'stafsscheduling', 'time_dimension', 'Trigger Functions', 'Types', and 'Views'.
- Dashboard:** Shows tabs for 'Dependencies', 'Dependents', 'Processes', and 'Project 1 - Healthcare Database/postgres@IE 6750 Project 1'.
- Query:** Shows the query `Select * from equipment`.
- Data Output:** Displays the results of the query in a table format.

equipment_id	equipment_type	department_id	maintenance_schedule	equipment_name
1	CT Scanner	10	2024-10-08	Equipment 1
2	ECG Machine	10	2024-10-25	Equipment 2
3	ECG Machine	8	2024-09-29	Equipment 3
4	X-Ray Machine	6	2024-10-07	Equipment 4
5	Ultrasound	7	2024-10-24	Equipment 5
6	ECG Machine	7	2024-10-05	Equipment 6
7	CT Scanner	4	2024-10-28	Equipment 7
8	ECG Machine	3	2024-10-22	Equipment 8
9	MRI Machine	10	2024-10-24	Equipment 9
10	MRI Machine	10	2024-10-21	Equipment 10

Total rows: 10 of 10 Query complete 00:00:00.217 Ln 2, Col 1

- Patient Readmissions Fact_table

The screenshot shows a PostgreSQL database interface with the following details:

- Object Explorer:** Shows a tree view of tables, including 'Rules', 'Triggers', 'insurance_dimension', 'insuranceproviders', 'laboratorytests', 'medicalcodes', 'medicalprocedures', 'medicalprocedures_cleaned', 'medications', 'medicationsadministered', 'patient_dimension', 'patient_readmissions_fact' (selected), 'patient_readmissions_fact_cleaned', 'patientfeedback', 'patients', 'procedure_dimension', 'Columns (4)', and 'Columns (7)'.
- Dashboard:** Shows tabs for 'Dependencies', 'Dependents', 'Processes', and 'Project 1 - Healthcare Database/postgres@IE 6750 Project 1'.
- Query:** Shows the query `select * from patient_readmissions_fact`.
- Data Output:** Displays the results of the query in a table format.

readmission_id	patient_id	provider_id	admission_date	discharge_date	code_id	total_charges	satisfaction_score	associated_costs	
1	16292	16292	1479	20200810	20200810	222	17005.00	3	2104.00
2	24834	24834	1480	20201218	20201218	222	6945.00	3	668.00
3	24847	24847	1480	20200728	20200728	222	10102.00	3	2905.00
4	24905	24905	1479	20200822	20200822	222	11287.00	3	2874.00
5	24984	24984	1337	20200606	20200606	222	0.00	3	4809.00
6	24993	24993	1404	20201216	20201216	222	0.00	3	2471.00
7	110459	10459	1517	20200523	20200523	222	16444.00	3	2576.00
8	110488	10488	1425	20200915	20200915	222	10472.00	3	4394.00
9	110500	10500	1338	20201206	20201206	222	15337.00	3	2826.00
10	168830	18830	1330	20200302	20200306	222	8267.00	3	1070.00
11	168836	18836	1282	20200808	20200418	222	7800.00	3	4515.00
12	168846	18846	1403	20201126	20200728	222	14908.00	3	1843.00
13	168852	18852	1525	20200421	20200620	222	13250.00	3	2893.00
14	168858	18858	1344	20201128	20200626	222	14424.00	3	2977.00
15	168869	18869	1454	20201011	20200521	222	16841.00	3	2550.00
16	168917	18917	1440	20200227	20200501	222	18695.00	3	2942.00
17	168919	18919	1464	20200922	20201120	228	12955.00	3	3816.00
18	168971	18971	1349	20200414	20200726	774	13791.00	3	3600.00

Total rows: 1000 of 15499 Query complete 00:00:00.138 Ln 1, Col 40

Final Project 1 Report

- Patient Readmissions Fact_table_cleaned

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** Shows the database schema with tables like Rules, Triggers, insurance_dimension, laboratorytests, medicalcodes, medicalprocedures, medicalprocedures_cleaned, medications, medicationsadministered, patient_dimension, and patient.readmissions_fact_cleaned.
- Query Editor:** Contains the SQL command: `select * from patient_readmissions_fact_cleaned`.
- Data Output:** Displays the results of the query, which consists of 1000 rows of data from the patient.readmissions_fact_cleaned table. The columns include readmission_id, patient_id, provider_id, admission_date, discharge_date, code_id, total_charges, satisfaction_score, and associated_costs.

	readmission_id	patient_id	provider_id	admission_date	discharge_date	code_id	total_charges	satisfaction_score	associated_costs
1	110923	10923	1	2020-06-14	2020-10-15	222	10500.00	3	1265.00
2	110924	10924	1	2020-12-02	2020-07-15	222	7326.00	3	797.00
3	110925	10925	1	2020-05-23	2020-10-24	229	7627.00	3	3597.00
4	110926	10926	1	2020-05-27	2020-03-20	224	5940.00	3	4541.00
5	110927	10927	1	2020-07-02	2020-08-07	223	19080.00	3	3830.00
6	110928	10928	1	2020-09-05	2020-06-17	226	18394.00	3	3223.00
7	110929	10929	1	2020-08-02	2020-04-10	222	12863.00	3	3611.00
8	110930	10930	1	2020-06-12	2020-05-07	222	12799.00	3	4232.00
9	110931	10931	1	2020-04-14	2020-01-22	224	5695.00	3	2504.00
10	110932	10932	1	2020-05-22	2020-07-18	222	10891.00	3	1382.00
11	110933	10933	1	2020-08-23	2020-11-13	222	15832.00	3	3848.00
12	110934	10934	1	2020-09-06	2020-09-19	224	7410.00	3	1047.00
13	110935	10935	1	2020-01-22	2020-05-24	224	14362.00	3	2761.00
14	110936	10936	1	2020-01-25	2020-12-05	224	12834.00	3	3257.00
15	110937	10937	1	2020-01-15	2020-11-06	222	13221.00	3	3972.00
16	110938	10938	1	2020-01-16	2020-01-08	224	13572.00	3	2788.00
17	110939	10939	1	2020-03-01	2020-05-07	227	14891.00	3	3783.00
18	110940	10940	1	2020-03-23	2020-04-18	224	16579.00	3	3739.00

Total rows: 1000 of 275000 Query complete 00:00:00.233 Ln 1, Col 48

10. Geographic dimension Cleaned

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** Shows the database schema with tables like Constraints, Indexes, RLS Policies, Rules, Triggers, and geographic_dimension.
- Query Editor:** Contains the SQL command: `select * from geographic_dimension_cleaned`.
- Data Output:** Displays the results of the query, which consists of 23166 rows of data from the geographic_dimension_cleaned table. The columns include location_id, city, and state.

	location_id	city	state
1	15	20969	NH
2	16	41466	MH
3	17	94918	NH
4	18	86718	DC
5	19	87448	MS
6	20	66044	WA
7	21	74546	SC
8	22	70193	HI
9	23	64864	AK
10	24	28719	NC
11	25	56848	ND
12	26	47576	GU
13	27	97656	NV
14	28	17803	MO
15	29	55153	AL
16	30	67934	CO
17	31	84054	MD
18	32	65264	GA
19	33	67057	OR

Total rows: 1000 of 23166 Query complete 00:00:00.137 Ln 1, Col 43

Final Project 1 Report

11. Healthcare Providers

Object Explorer

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Query Query History

```
1 select * from healthcareproviders
```

Data Output Messages Notifications

SQL

	provider_id	first_name	last_name	specialty	department_id	contact_info
1	1	Marie	Contreras	Neurology	[null]	(437)226-2651
2	2	Thomas	Spears	Neurology	[null]	(845)262-7081x96335
3	3	Lisa	Young	Orthopedics	[null]	(641)914-6955x47035
4	4	Audrey	Lin	Emergency Medicine	[null]	956.348.4260
5	5	Deborah	Zimmerman	Pediatrics	[null]	711.671.3887x4853
6	6	Shaun	Shaw	Emergency Medicine	[null]	(692)468-8211x7417
7	7	Daniel	Davis	Oncology	[null]	(606)680-0669x674
8	8	Charles	Anderson	Oncology	[null]	001-373-575-4035x8119
9	9	Karen	Wilson	Emergency Medicine	[null]	001-912-895-9213
10	10	Gary	Werner	Oncology	[null]	+1-720-387-8690x38645
11	11	Christopher	Holmes	Emergency Medicine	[null]	983-490-5677
12	12	David	Johnson	Orthopedics	[null]	713-810-5416x175
13	13	Destiny	Bryant	Emergency Medicine	[null]	(288)530-6540
14	14	Amy	Anderson	Pediatrics	[null]	356-766-7764
15	15	Tammy	White	Oncology	[null]	447-242-5068x9240
16	16	Krista	Richardson	Oncology	[null]	(342)338-5151
17	17	Megan	Williams	Neurology	[null]	+1-794-521-1740x2717
18	18	Evan	Thornton	Pediatrics	[null]	001-452-354-6414x79280

Total rows: 1000 of 1640 Query complete 00:00:00.258 Ln 1, Col 34

12. Healthcare Providers_cleaned

Object Explorer

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Query Query History

```
1 select * from healthcareproviders_cleaned
```

Data Output Messages Notifications

SQL

	provider_id	first_name	last_name	specialty	department_id	contact_info
1	581	Ryan	Guerrero	Cardiology	2	993-526-0815x463
2	603	Kayla	Phillips	Emergency Medicine	7	443-250-9902
3	604	Scott	Marsh	Emergency Medicine	7	270.265.9887x2023
4	605	Katrina	Huffman	Neurology	9	001-660-309-4702x45798
5	606	Austin	Jordan	Orthopedics	10	001-557-271-1042
6	607	Brittany	Rivera	Neurology	9	001-728-969-1197
7	608	Shawn	Nguyen	Emergency Medicine	7	974-549-2773x889
8	609	Randy	Griffin	Neurology	9	001-304-671-9311
9	610	Denise	Robertson	Neurology	9	(288)865-1531x05565
10	611	Dennis	Franco	Pediatrics	12	001-947-511-9681x0793
11	612	Shawn	Juarez	Cardiology	8	+1-914-996-8479x2767
12	613	Thomas	Floyd	Neurology	9	+1-579-636-5625x693
13	614	Brian	Clark	Oncology	11	482-524-8486
14	615	Robert	Smith	Neurology	9	+1-832-575-6408
15	616	Heidi	Gonzalez	Orthopedics	10	+1-594-933-4535x6694
16	617	Anthony	Hamilton	Pediatrics	12	+1-319-714-8448
17	618	Debra	Church	Neurology	9	(398)240-7236
18	619	Brittany	Robinson	Emergency Medicine	7	001-283-354-0556

Final Project 1 Report

13. Hospital_Departments

Object Explorer

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Query Query History

```
1 select * from hospitaldepartments
2
```

Data Output Messages Notifications

	department_id	department_name	location	services_offered
1	1	Emergency	First Floor	Emergency care services
2	2	Cardiology	Second Floor	Heart-related treatments
3	3	Neurology	Third Floor	Brain and nervous system services
4	4	Orthopedics	Second Floor	Bone and muscle treatments
5	5	Oncology	Fourth Floor	Cancer treatments
6	6	Pediatrics	First Floor	Child healthcare services
7	7	Emergency	First Floor	Emergency care services
8	8	Cardiology	Second Floor	Heart-related treatments
9	9	Neurology	Third Floor	Brain and nervous system services
10	10	Orthopedics	Second Floor	Bone and muscle treatments
11	11	Oncology	Fourth Floor	Cancer treatments
12	12	Pediatrics	First Floor	Child healthcare services
13	13	Emergency	First Floor	Emergency care services
14	14	Cardiology	Second Floor	Heart-related treatments
15	15	Neurology	Third Floor	Brain and nervous system services
16	16	Orthopedics	Second Floor	Bone and muscle treatments
17	17	Oncology	Fourth Floor	Cancer treatments
18	18	Pediatrics	First Floor	Child healthcare services

Total rows: 204 of 204 Query complete 00:00:00.109 Ln 2, Col 1

14. Insurance_Providers

Object Explorer

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Query Query History

```
1 select * from insuranceproviders
2
```

Data Output Messages Notifications

	insurance_provider_id	provider_name	contact_info
1	1	Anderson, Mullins and Young	33970 Ralph Hollow Apt. 885
2	2	Nichols-Meyer	3023 Benjamin Fork Apt. 571
3	3	Doyle, Walker and Hurn	7282 Jerry Villages
4	4	Romero, Cook and Lynch	460 David Well Suite 138
5	5	Ruiz Gordon	2885 Kenneth Bypass Suite 094
6	6	Patel, Johnson and Davis	22287 Harris Villa
7	7	Thomas, Robertson and Anderson	5998 Tyler Mountains
8	8	Lee, Padilla and Brown	15010 Maya Locks
9	9	Nelson and Sons	0034 Dakota Rue
10	10	Rose, White and Davis	949 Andre Field Suite 657
11	11	Smith, Myers and Kelly	030 Edwards Plain Apt. 521
12	12	Ryan LLC	8460 Fischer Valleys Suite 428
13	13	Cruz, Maldonado and Jensen	95492 Castro Inlet Apt. 888
14	14	Fernandez, Brown and Jones	5657 David Point
15	15	Mccarthy Group	1419 Bryan Square
16	16	Hampton Ltd	30293 Jeffrey Cliff Apt. 307
17	17	Thornton, Anderson and Johnson	38419 Ortiz Light Suite 847
18	18	Parker-Mendoza	48153 Tammy Lake Suite 729

Total rows: 880 of 880 Query complete 00:00:00.094 Ln 1, Col 33

Final Project 1 Report

15. Insurance dimension

Object Explorer

```

> Rules
> Triggers
> discharges
> equipment
> geographic_dimension
> geographic_dimension_cleaned
> healthcareproviders
> healthcareproviders_cleaned
> hospitaldepartments
> Columns (4)
  department_id
  department_name
  location
  services_offered
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
> insurance_dimension
> insuranceproviders
> Columns (3)
  insurance_provider_id
  provider_name
  contact_info
> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
> laboratorytests
> medicalcodes
> medicalprocedures
> medicalprocedures_cleaned
> medications
> medicationsadministered
> patient_dimension
> patient_readmissions_fact

```

Dashboard Dependencies Dependents Processes Project 1 - Healthcare Database/postgres@IE 6750 Project 1

Query Query History

```

1 select * from insurance_dimension
2

```

Data Output Messages Notifications

	insurance_provider_id	provider_name	contact_info
1	1	Anderson, Mullins and Young	33970 Ralph Hollow Apt. 885
2	2	Nichols-Meyer	3023 Benjamin Fork Apt. 571
3	3	Doyle, Walker and Hunt	7262 Jerry Villages
4	4	Romero, Cook and Lynch	460 David Well Suite 138
5	5	Ruiz-Gordon	2885 Kenneth Bypass Suite 094
6	6	Patel, Johnson and Davis	22287 Harris Via
7	7	Thomas, Robertson and Anderson	5998 Tyler Mountains
8	8	Lee, Padilla and Brown	15010 Mayo Locks
9	9	Nelson and Sons	0034 Dakota Rue
10	10	Rose, White and Davis	949 Andre Field Suite 657
11	11	Smith, Myers and Kelly	030 Edwards Plain Apt. 521
12	12	Ryan LLC	8460 Fischer Valley Suite 428
13	13	Cruz, Maldonado and Jensen	95492 Castro Inlet Apt. 888
14	14	Fernandez, Brown and Jones	5657 David Point
15	15	McCarthy Group	1419 Bryan Square
16	16	Hampton Ltd	30293 Jeffrey Cliff Apt. 307
17	17	Thornton, Anderson and Johnson	38419 Ortiz Light Suite 847
18	18	Rodriguez-Mendoza	48153 Tammy Lake Suite 729

Servers > IE 6750 Project 1 > Databases > Project 1 - Healthcare Database > Schemas > public > Tables > equipment. Implete 00:00:00.140 Ln 1, Col 34

16. Laboratory Tests

Object Explorer

```

> Constraints
> Indexes
> RLS Policies
> Rules
> Triggers
> laboratorytests
> medicalcodes
> medicalprocedures
> medicalprocedures_cleaned
> medications
> medicationsadministered
> patient_dimension
> patient_readmissions_fact
> patient_readmissions_fact_cleaned
> patientfeedback
> patients
> procedure_dimension
> provider_dimension
> readmissionrecords
> staffscheduling
> time_dimension
> Trigger Functions
> Types
> Views
> Subscriptions
> postgres
> Login/Group Roles
> Tablespaces (2)

```

Dashboard Dependencies Dependents Processes Project 1 - Healthcare Database/postgres@IE 6750 Project 1

Query Query History

```

1 select * from laboratorytests
2

```

Data Output Messages Notifications

	lab_test_id	admission_id	test_type	test_code	test_date	result
1	1	675001	Blood Panel	7225	2020-09-09	Abnormal
2	2	675002	Glucose	3389	2020-04-12	Abnormal
3	3	675003	Blood Panel	2554	2020-06-02	Normal
4	4	675004	Glucose	7059	2020-10-21	Normal
5	5	675005	Blood Panel	2352	2020-05-29	Normal
6	6	675006	A1C	9223	2020-10-07	Normal
7	7	675007	Blood Panel	6099	2020-01-04	Normal
8	8	675008	Blood Panel	7606	2020-03-30	Abnormal
9	9	675009	Lipid Panel	4552	2020-10-21	Normal
10	10	675010	A1C	5317	2020-05-26	Normal
11	11	675011	A1C	3443	2020-08-19	Abnormal
12	12	675012	A1C	4467	2020-01-21	Normal
13	13	675013	Blood Panel	5027	2020-07-04	Abnormal
14	14	675014	Lipid Panel	6788	2020-06-28	Normal
15	15	675015	Glucose	6528	2020-12-02	Normal
16	16	675016	A1C	9928	2020-06-09	Normal
17	17	675017	Glucose	5094	2020-10-04	Normal
18	18	675018	Rinno Panel	1572	2020-05-30	Abnormal

Total rows: 1000 of 300000 Query complete 00:00:00.458 Ln 1, Col 30

Final Project 1 Report

17. Medical Codes

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** On the left, under the "insuranceprovider" node, the "medicalcodes" table is selected.
- Query Editor:** The query `select * from medicalcodes` is run, resulting in 324 rows.
- Data Output:** The results are displayed in a table with columns: code_id [PK] integer, code_type character varying(20), code_value character varying(20), and description text.
- Table Data:** The table contains rows 1 through 18, with descriptions such as "Type 2 diabetes mellitus without complications" and "Essential (primary) hypertension".

18. Medical Procedures

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** On the left, under the "insuranceprovider" node, the "medicalprocedures" table is selected.
- Query Editor:** The query `select * from medicalprocedures` is run, resulting in 1000 rows.
- Data Output:** The results are displayed in a table with columns: procedure_record_id [PK] integer, admission_id integer, code_id integer, procedure_date date, provider_id integer, and procedure_outcome character varying(255).
- Table Data:** The table contains rows 1 through 18, all marked as "Successful".

Final Project 1 Report

19. Medical Procedures Cleaned

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** On the left, under the "hospitaldepartments" schema, the "medicalprocedures_cleaned" table is selected.
- Query Editor:** The main area displays a SQL query: `select * from medicalprocedures_cleaned`. The results show 18 rows of data.
- Data Output:** The results are presented in a table with columns: procedure_record_id, admission_id, code_id, procedure_date, provider_id, and procedure_outcome.

	procedure_record_id	admission_id	code_id	procedure_date	provider_id	procedure_outcome
1	478766	708180	223	2020-10-25	1	Successful
2	478767	708180	222	2020-10-25	1	Successful
3	478768	708181	222	2020-12-17	1	Successful
4	478769	708181	222	2020-12-17	1	Successful
5	478770	708181	222	2020-12-17	1	Successful
6	478771	708182	228	2020-02-19	1	Successful
7	478772	708182	223	2020-02-19	1	Successful
8	478773	708182	222	2020-02-19	1	Successful
9	478774	708183	224	2020-06-15	1	Successful
10	478775	708183	224	2020-06-15	1	Successful
11	478776	708183	223	2020-06-15	1	Successful
12	478777	708184	228	2020-07-29	1	Successful
13	478778	708184	224	2020-07-29	1	Successful
14	478779	708184	223	2020-07-29	1	Successful
15	478780	708185	226	2020-09-01	1	Successful
16	478781	708185	224	2020-09-01	1	Successful
17	478782	708185	224	2020-09-01	1	Successful
18	478783	708186	223	2020-05-18	1	Successful

Total rows: 1000 of 1050000 Query complete 00:00:00.614 Ln 1, Col 40

20. Medications

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** On the left, under the "hospitaldepartments" schema, the "medications" table is selected.
- Query Editor:** The main area displays a SQL query: `select * from medications`. The results show 18 rows of data.
- Data Output:** The results are presented in a table with columns: medication_id, generic_name, brand_name, dosage_form, and manufacturer.

	medication_id	generic_name	brand_name	dosage_form	manufacturer
1	1	Atorvastatin	Lipitor	Tablet	Pfizer
2	2	Levothyroxine	Synthroid	Tablet	AbbVie
3	3	Lisinopril	Prinivil	Tablet	Merck
4	4	Metformin	Glicophage	Tablet	Bristol-Myers Squibb
5	5	Amiodipine	Norvasc	Tablet	Pfizer
6	6	Metoprolol	Lopressor	Tablet	Novartis
7	7	Atorvastatin	Lipitor	Tablet	Pfizer
8	8	Levothyroxine	Synthroid	Tablet	AbbVie
9	9	Lisinopril	Prinivil	Tablet	Merck
10	10	Metformin	Glicophage	Tablet	Bristol-Myers Squibb
11	11	Amiodipine	Norvasc	Tablet	Pfizer
12	12	Metoprolol	Lopressor	Tablet	Novartis
13	13	Atorvastatin	Lipitor	Tablet	Pfizer
14	14	Levothyroxine	Synthroid	Tablet	AbbVie
15	15	Lisinopril	Prinivil	Tablet	Merck
16	16	Metformin	Glicophage	Tablet	Bristol-Myers Squibb
17	17	Amiodipine	Norvasc	Tablet	Pfizer
18	18	Metoprolol	Lopressor	Tablet	Novartis

Total rows: 480 of 480 Query complete 00:00:00.162 Ln 2, Col 1

Final Project 1 Report

21. Medications Administered

The screenshot shows the Object Explorer on the left with the 'medicationsadministered' table highlighted. The main area displays a SQL query and its results.

```
select * from medicationsadministered
```

medication_admin_id	admission_id	medication_id	dosage	route_of_administration	administration_date
1	1	650001	1 tablet(s)	Oral	2020-11-14
2	2	650002	4 tablets(s)	Oral	2020-11-23
3	3	650003	2 tablets(s)	Oral	2020-11-14
4	4	650004	4 tablets(s)	Oral	2020-12-09
5	5	650005	6 tablets(s)	Oral	2020-02-15
6	6	650006	1 tablet(s)	Oral	2020-10-31
7	7	650007	4 tablets(s)	Oral	2020-11-27
8	8	650008	3 tablets(s)	Oral	2020-10-06
9	9	650009	2 tablets(s)	Oral	2020-09-16
10	10	650010	1 tablet(s)	Oral	2020-03-19
11	11	650011	6 tablets(s)	Oral	2020-01-22
12	12	650012	2 tablets(s)	Oral	2020-01-04
13	13	650013	3 tablets(s)	Oral	2020-09-10
14	14	650014	5 tablets(s)	Oral	2020-04-22
15	15	650015	1 tablet(s)	Oral	2020-10-01
16	16	650016	4 tablets(s)	Oral	2020-06-20
17	17	650017	1 tablet(s)	Oral	2020-04-13
18	18	650018	6 tablets(s)	Oral	2020-12-11

Total rows: 1000 of 325000 Query complete 00:00:00.455 Ln 1, Col 38

22. Patient dimension

The screenshot shows the Object Explorer on the left with the 'patient_dimension' table highlighted. The main area displays a SQL query and its results.

```
select * from patient_dimension
```

patient_id	age_group	gender	insurance_provider_id
1829	71-80	Male	848
1830	90+	Male	754
2031	61-70	Female	426
2032	61-70	Female	498
2033	71-80	Male	474
2034	81-90	Female	108
2035	81-90	Male	608
2037	71-80	Female	742
2038	71-80	Female	11
2039	71-80	Female	397
2040	51-60	Female	343
2041	90+	Female	183
2042	81-90	Female	84
2043	61-70	Male	388
2044	81-90	Female	548
2045	41-50	Male	43
2046	81-90	Male	532
2047	61-70	Female	676

Total rows: 1000 of 25001 Query complete 00:00:00.164 Ln 1, Col 32

Final Project 1 Report

23. Patient Feedback

Object Explorer

- > insurance_dimension
- > insuranceproviders
- > laboratorytests
- > medicalcodes
- > medicalprocedures
- > medicalprocedures_cleaned
- > medications
- > medicationsadministered
- > patient_dimension
- > patient_readmissions_fact
- > patient_readmissions_fact_cleaned
- > patientfeedback
 - > Columns (4)
 - feedback_id
 - admission_id
 - satisfaction_score
 - comments
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- > patients
 - > Columns (7)
 - patient_id
 - first_name
 - last_name
 - date_of_birth
 - gender
 - contact_info
 - insurance_provider_id
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- > procedure_dimension
- > provider_dimension
- > readmissionrecords

Dashboard > Dependencies > Dependents > Processes > Project 1 - Healthcare Database/postgres@IE 6750 Project 1 >

Query Query History

1 select * from patientfeedback

	feedback_id	admission_id	satisfaction_scores	comments
	[PK] integer	integer	integer	text
1	1	725001	1	Commercial today save scene yourself event very daughter majority court.
2	2	725002	2	Democrat difference federal measure than carry common.
3	3	725003	4	Else worse because pressure large painting mention.
4	4	725004	1	East capital age book buy order same remember.
5	5	725005	3	Ago season customer clear manager modern half represent reality box artist.
6	6	725006	1	Cell yeah provide buy billion hour power.
7	7	725007	3	Follow whatever old identify for focus sound.
8	8	725008	2	Up food do cause with sing break team newspaper.
9	9	725009	2	Officer step forget whatever baby staff near participant front girl next key site.
10	10	725010	5	Easy them clearly herself by industry knowledge difference catch recently.
11	11	725011	2	Quality second appear matter clear could sure son.
12	12	725012	5	Cultural big author human risk show close home star that attention guess.
13	13	725013	5	Director attack goal five i.n.
14	14	725014	3	Region fast explain response interview let foot ago within behind.
15	15	725015	2	Focus bed seek soon increase conference now blue section perform us sister draw.
16	16	725016	5	Represent institution without nearly rule learn.
17	17	725017	1	Not democratic agreement way up note.
18	18	725018	3	Discover those situation institution different relentless.

Total rows: 1000 of 250000 Query complete 00:00:00.403 Ln 1, Col 23

24. Patients

Object Explorer

- > insurance_dimension
- > insuranceproviders
- > laboratorytests
- > medicalcodes
- > medicalprocedures
- > medicalprocedures_cleaned
- > medications
- > medicationsadministered
- > patient_dimension
- > patient_readmissions_fact
- > patient_readmissions_fact_cleaned
- > patientfeedback
 - > Columns (4)
 - feedback_id
 - admission_id
 - satisfaction_score
 - comments
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- > patients
 - > Columns (7)
 - patient_id
 - first_name
 - last_name
 - date_of_birth
 - gender
 - contact_info
 - insurance_provider_id
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- > procedure_dimension
- > provider_dimension
- > readmissionrecords

Dashboard > Dependencies > Dependents > Processes > Project 1 - Healthcare Database/postgres@IE 6750 Project 1 >

Query Query History

1 select * from patients

	patient_id	first_name	last_name	date_of_birth	gender	contact_info	insurance_provider_id
	[PK] integer	character varying	character varying	date	character varying (10)	text	integer
1	2032	Jessie	Stewart	1959-01-01	Female	4114 Sutton Lakes	498
2	2033	Emerson	Turner	1949-01-01	Male	Unit 0725 Box 2711	474
3	2483	Toby	Bailey	1929-01-01	Female	USS Reyes	30
4	2698	Sam	Hill	1949-01-01	Male	USCGC Gray	120
5	2034	Alex	Allen	1939-01-01	Female	PSC 2256, Box 6198	108
6	2035	Toby	Ramos	1939-01-01	Male	1049 McMillan Green	608
7	2037	Mason	Wright	1949-01-01	Female	832 Jean Course Suite 708	742
8	2947	Harper	Stewart	1939-01-01	Male	USNV Garcia	809
9	2038	Chandler	Robinson	1949-01-01	Female	1237 Lozano Orchard Suite 424	11
10	3107	Dak	Campbell	1969-01-01	Male	USS Byrd	167
11	3304	Chris	Rodriguez	1949-01-01	Male	USNV Dennis	741
12	3828	Riley	Evans	1959-01-01	Female	Unit 9926 Box 9112	729
13	3996	Shay	Baker	1969-01-01	Male	USS Herrera	522
14	2039	Jaden	Allen	1949-01-01	Female	78634 Berry Loft	397
15	4125	Kai	Miller	1969-01-01	Male	USNS Wong	430
16	4877	Val	Ramos	1939-01-01	Female	USNV Petty	185
17	5053	Corey	Lee	1969-01-01	Female	PSC 0201, Box 4463	753
18	7040	Chandler	Hill	1949-01-01	Female	2131 Castillo Ridge	341

Total rows: 1000 of 250000 Query complete 00:00:00.135 Ln 1, Col 23

Final Project 1 Report

25. Provider dimension

Object Explorer

```
admission_date
discharge_date
code_id
total_charges
satisfaction_score
associated_costs
Constraints (5)
    patient_readmissions_fact_cleaned_code_id_fkey
    patient_readmissions_fact_cleaned_patient_id_fkey
    patient_readmissions_fact_cleaned_pkkey
    patient_readmissions_fact_cleaned_provider_id_fkey
    patient_readmissions_provider_id_fkey
Indexes
RLS Policies
Rules
Triggers
patientfeedback
patients
provider_dimension
    Columns (3)
        provider_id
        specialty
        department_id
    Constraints (5)
    Indexes
    RLS Policies
    Rules
    Triggers
    readmissionrecords
    staffscheduling
    time_dimension
    Trigger Functions
    Types
    Views
    Subscriptions
    postgres
    Login/Group Roles
Tablespaces (2)
    pg_default
```

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Project 1 - Healthcare Database/postgres@IE 6750 Project 1

No limit ▾

Query History

1 Select * from provider_dimension

Data Output Messages Notifications

	provider_id	specialty	department_id
1	581	Cardiology	2
2	582	Orthopedics	4
3	583	Emergency Medicine	1
4	584	Pediatrics	6
5	585	Cardiology	2
6	586	Emergency Medicine	1
7	587	Orthopedics	4
8	588	Oncology	5
9	589	Orthopedics	4
10	590	Neurology	3
11	591	Orthopedics	4
12	592	Orthopedics	4
13	593	Cardiology	2
14	594	Neurology	3
15	595	Cardiology	2
16	596	Orthopedics	4
17	597	Neurology	3
18	598	Pediatrics	6

Total rows: 1000 of 1060 Query complete 00:00:00.121 Ln 1, Col 33

26. Readmission Records

Object Explorer

```
patient_dimension
patient_readmissions_fact
patient_readmissions_fact_cleaned
    Columns (9)
        readmission_id
        patient_id
        provider_id
        admission_date
        discharge_date
        code_id
        total_charges
        satisfaction_score
        associated_costs
    Constraints (5)
        patient_readmissions_fact_cleaned_code_id_fkey
        patient_readmissions_fact_cleaned_patient_id_fkey
        patient_readmissions_fact_cleaned_pkkey
        patient_readmissions_fact_cleaned_provider_id_fkey
        patient_readmissions_provider_id_fkey
Indexes
RLS Policies
Rules
Triggers
patientfeedback
patients
provider_dimension
readmissionrecords
    Columns (5)
        readmission_id
        original_admission_id
        readmission_date
        readmission_reason
        associated_costs
    Constraints (5)
    Indexes
    RLS Policies
    Rules
    Triggers
    staffscheduling
```

Dashboard X Dependencies X Dependents X Processes X Project 1 - Healthcare Database/postgres@IE 6750 Project 1 * X

Project 1 - Healthcare Database/postgres@IE 6750 Project 1

No limit ▾

Query History

1 Select * from readmissionrecords

Execute script [F5]

Data Output Messages Notifications

	readmission_id	original_admission_id	readmission_date	readmission_reason	associated_costs
1	1	700001	2020-08-02	Wall kitchen next anyone.	2528.00
2	2	700002	2020-08-06	Although around heart they talk.	977.00
3	3	700003	2021-02-25	List count land forget perhaps family.	2454.00
4	4	700004	2020-04-02	Color first section edge worker.	2363.00
5	5	700005	2020-12-03	Everything set guy country media necessary prove.	3539.00
6	6	700006	2020-09-30	Turn law impact dog sign.	3681.00
7	7	700007	2020-02-15	Leave miss keep remain agent road.	1796.00
8	8	700008	2020-05-29	Choice Mrs read seem.	631.00
9	9	700009	2020-06-09	Century serious majority anything economy.	960.00
10	10	700010	2020-11-07	Pick leader everybody seem young these old mention.	3010.00
11	11	700011	2021-02-16	Truth station only mission people.	3925.00
12	12	700012	2020-05-02	Community few view cost rock listen.	2743.00
13	13	700013	2020-11-13	Attorney sport usually lead.	4034.00
14	14	700014	2020-06-08	City that picture your traditional likely defense event.	3415.00
15	15	700015	2020-11-07	Month traditional young leave business decision.	2307.00
16	16	700016	2020-11-26	Pressure generation part ready.	952.00
17	17	700017	2020-03-26	Order black street.	4031.00
18	18	700018	2021-07-27	Which offer low culture attack clear ever	4446.00

Total rows: 1000 of 275000 Query complete 00:00:00.265 Ln 1, Col 33

Final Project 1 Report

27. Staff Scheduling

Object Explorer

- patient_dimension
- patient_readmissions_fact
- patient_readmissions_fact_cleaned
 - Columns (9)
 - readmission_id
 - patient_id
 - provider_id
 - admission_date
 - discharge_date
 - code_id
 - total_charges
 - satisfaction_score
 - associated_costs
 - Constraints (5)
 - patient_readmissions_fact_cleaned_code_id_fkey
 - patient_readmissions_fact_cleaned_patient_id_fkey
 - patient_readmissions_fact_cleaned_pk
 - patient_readmissions_fact_cleaned_provider_id_fkey
 - patient_readmissions_provider_id_fkey
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
 - patientfeedback
 - patients
 - provider_dimension
 - readmissionrecords
 - staffscheduling
 - time_dimension
 - Trigger Functions
 - Types
 - Views
 - Subscriptions
 - postgres
 - Login/Group Roles
- Tablespaces (2)
 - pg_default
 - pg_global
- PostgreSQL 17

Dashboard × Dependencies × Dependents × Processes × Project 1 - Healthcare Database/postgres@IE 6750 Project 1 ×

Query Query History

```
1 Select * from staffscheduling
2
```

Data Output Messages Notifications

SQL

schedule_id	provider_id	shift_date	shift_start_time	shift_end_time	time without time zone	department_id
1	581	2024-09-06	10:27:13		13:38:55	2
2	582	2024-09-23	23:46:04		03:52:54	4
3	583	2024-09-23	17:56:06		23:28:16	1
4	584	2024-09-23	04:27:46		13:49:03	6
5	585	2024-09-13	18:51:49		20:12:37	2
6	586	2024-09-03	10:20:08		10:41:55	1
7	587	2024-09-10	23:14:16		02:49:11	4
8	588	2024-09-07	13:11:19		16:24:42	5
9	589	2024-09-19	17:18:19		10:42:37	4
10	590	2024-09-23	03:36:29		09:27:15	3
11	591	2024-09-17	20:02:25		17:46:57	4
12	592	2024-09-22	01:57:19		03:43:20	4
13	593	2024-09-19	13:11:43		22:05:00	2
14	594	2024-09-19	09:02:31		12:14:45	3
15	595	2024-09-23	08:10:52		15:38:00	2
16	596	2024-09-24	13:10:04		21:11:45	4
17	597	2024-09-04	13:38:16		07:49:45	3
18	598	2024-09-14	17:53:01		23:50:41	6

Total rows: 1000 of 7920 Query complete 00:00:00.112 Ln 2, Col 1

28. Time dimension

Object Explorer

- patient_dimension
- patient_readmissions_fact
- patient_readmissions_fact_cleaned
 - Columns (9)
 - readmission_id
 - patient_id
 - provider_id
 - admission_date
 - discharge_date
 - code_id
 - total_charges
 - satisfaction_score
 - associated_costs
 - Constraints (5)
 - patient_readmissions_fact_cleaned_code_id_fkey
 - patient_readmissions_fact_cleaned_patient_id_fkey
 - patient_readmissions_fact_cleaned_pk
 - patient_readmissions_fact_cleaned_provider_id_fkey
 - patient_readmissions_provider_id_fkey
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
 - patientfeedback
 - patients
 - provider_dimension
 - readmissionrecords
 - staffscheduling
 - time_dimension
 - Trigger Functions
 - Types
 - Views
 - Subscriptions
 - postgres
 - Login/Group Roles
- Tablespaces (2)
 - pg_default
 - pg_global
- PostgreSQL 17

Dashboard × Dependencies × Dependents × Processes × Project 1 - Healthcare Database/postgres@IE 6750 Project 1 ×

Query Query History

```
1 Select * from time_dimension
2
```

Data Output Messages Notifications

SQL

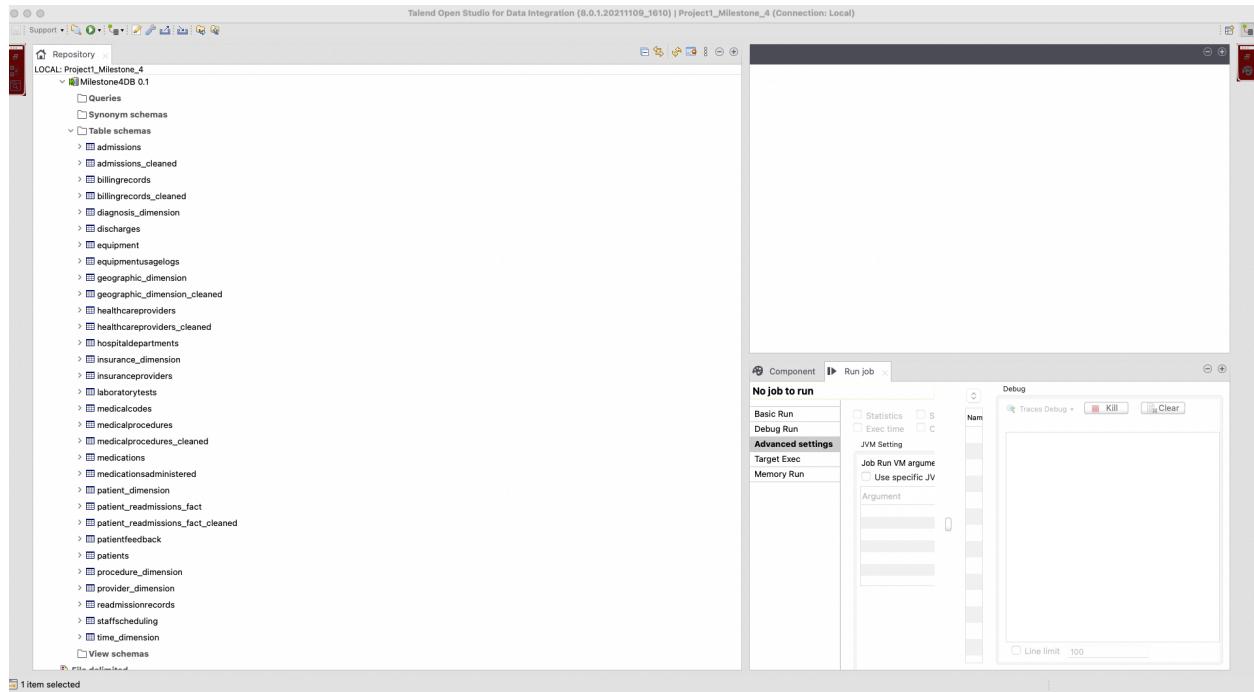
date_id	date	date_value	day	month	quarter	year	date_yyyymmdd
1	6	2020-01-06	6	1	1	2020	20200106
2	7	2020-01-07	7	1	1	2020	20200107
3	8	2020-01-08	8	1	1	2020	20200108
4	9	2020-01-09	9	1	1	2020	20200109
5	10	2020-01-10	10	1	1	2020	20200110
6	11	2020-01-11	11	1	1	2020	20200111
7	12	2020-01-12	12	1	1	2020	20200112
8	13	2020-01-13	13	1	1	2020	20200113
9	14	2020-01-14	14	1	1	2020	20200114
10	15	2020-01-15	15	1	1	2020	20200115
11	16	2020-01-16	16	1	1	2020	20200116
12	17	2020-01-17	17	1	1	2020	20200117
13	18	2020-01-18	18	1	1	2020	20200118
14	19	2020-01-19	19	1	1	2020	20200119
15	20	2020-01-20	20	1	1	2020	20200120
16	21	2020-01-21	21	1	1	2020	20200121
17	22	2020-01-22	22	1	1	2020	20200122
18	23	2020-01-23	23	1	1	2020	20200123

Total rows: 465 of 465 Query complete 00:00:00.107 Ln 1, Col 29

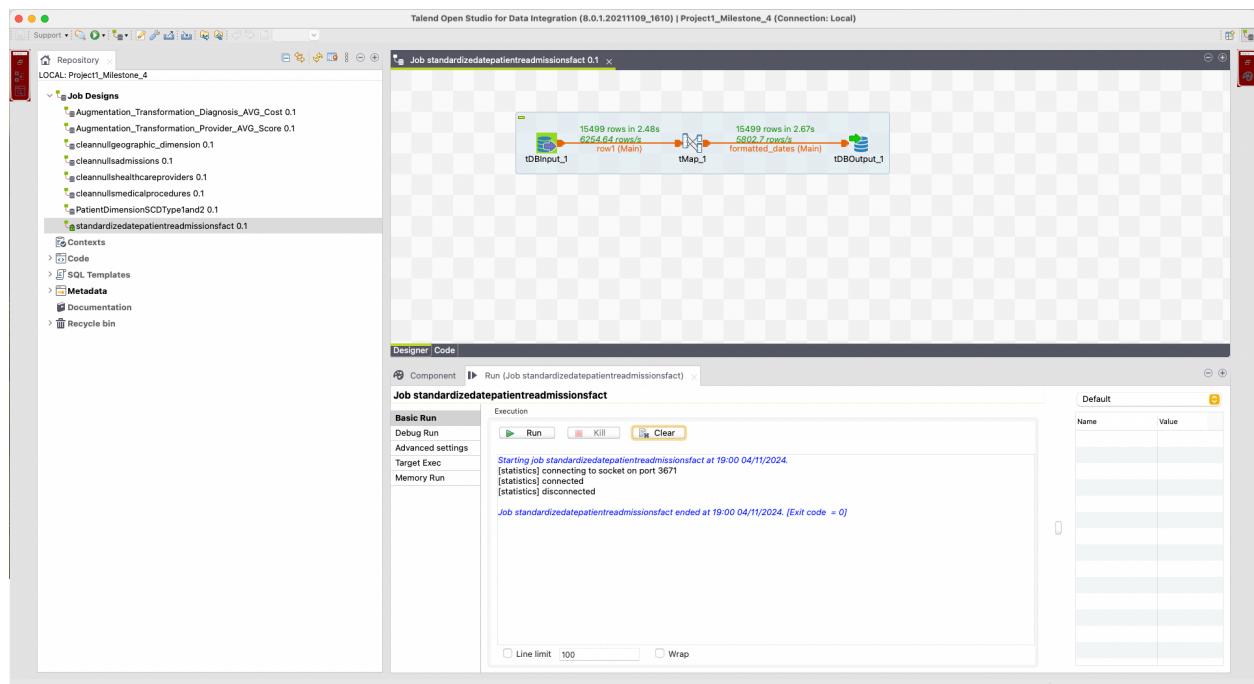
Final Project 1 Report

ii. Talend Implementation

- **Loaded Tables from Database**

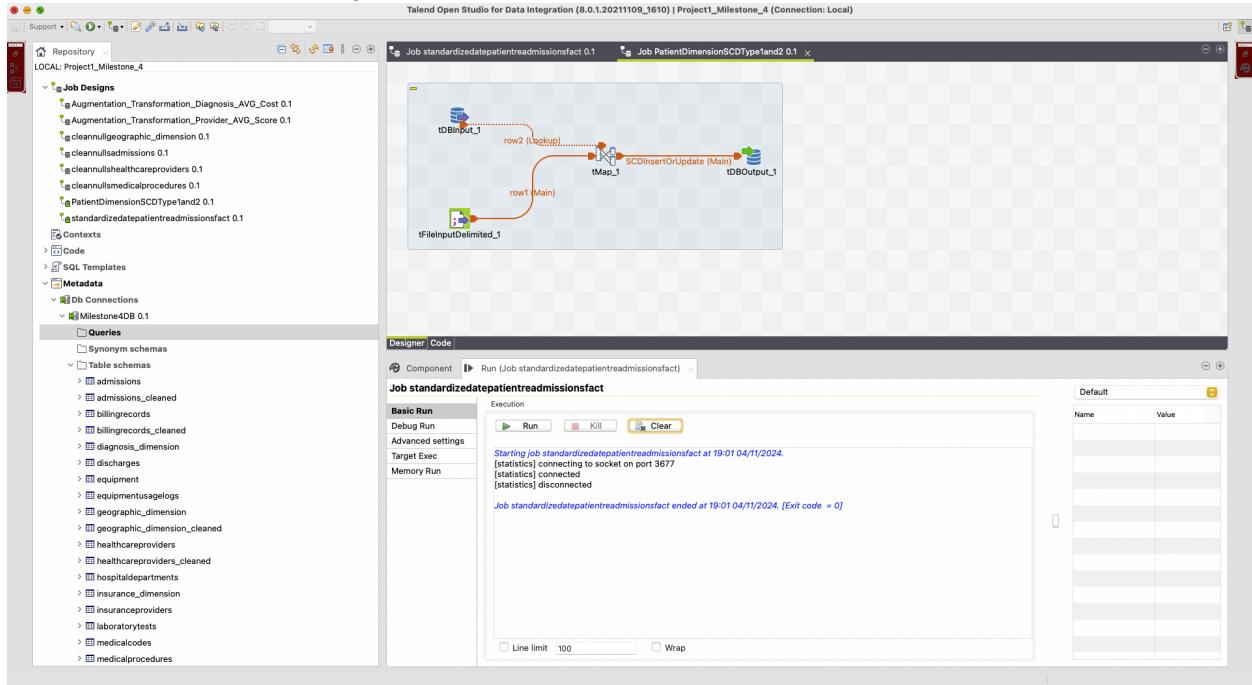


- **Standardized Patient Readmission Fact**

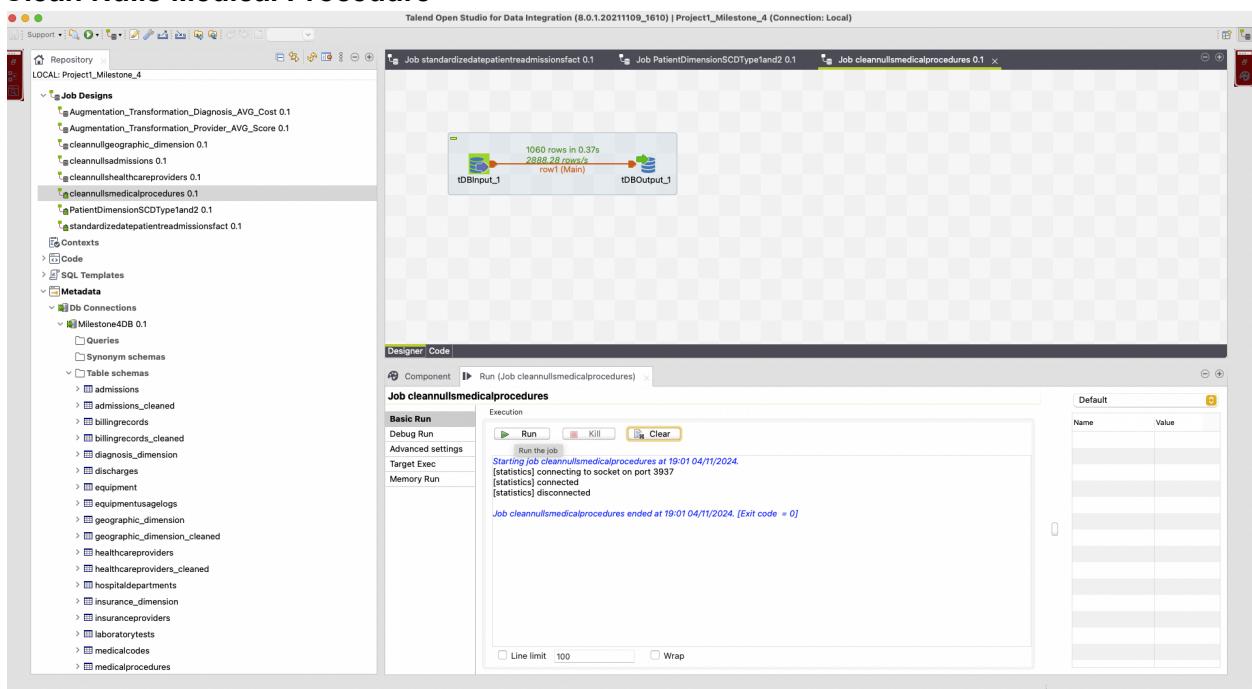


Final Project 1 Report

- Patient Dimension SCD Type 1 and 2

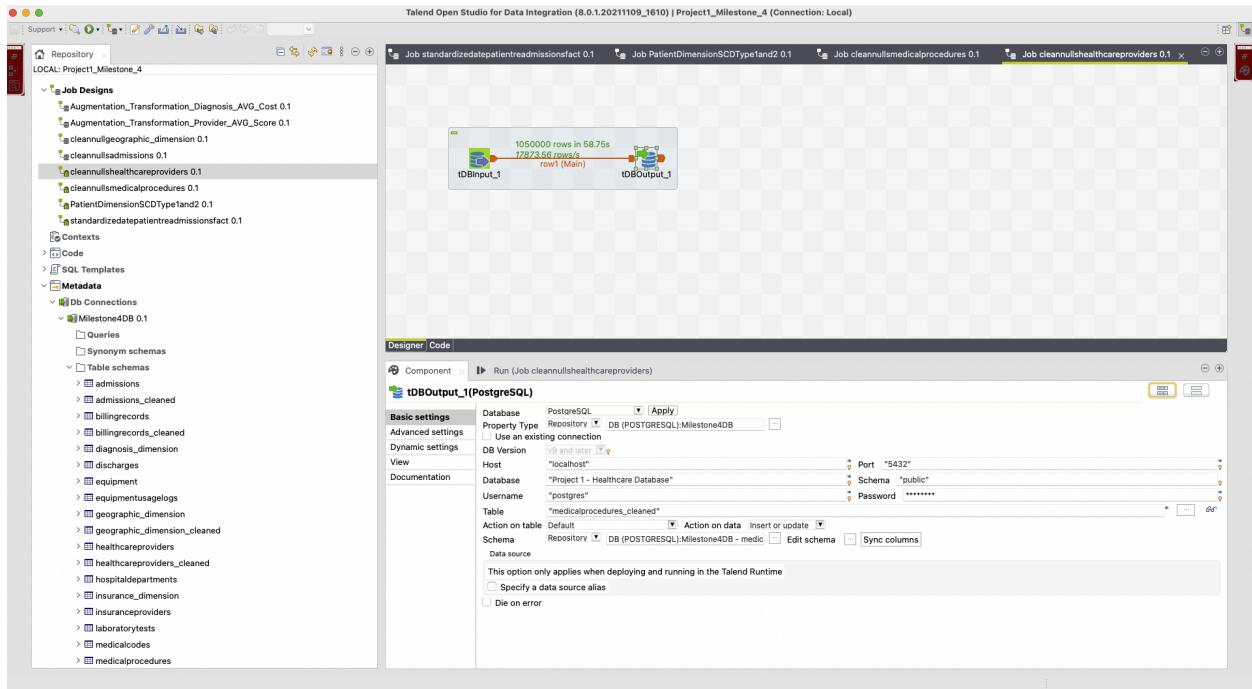


- Clean Nulls Medical Procedure

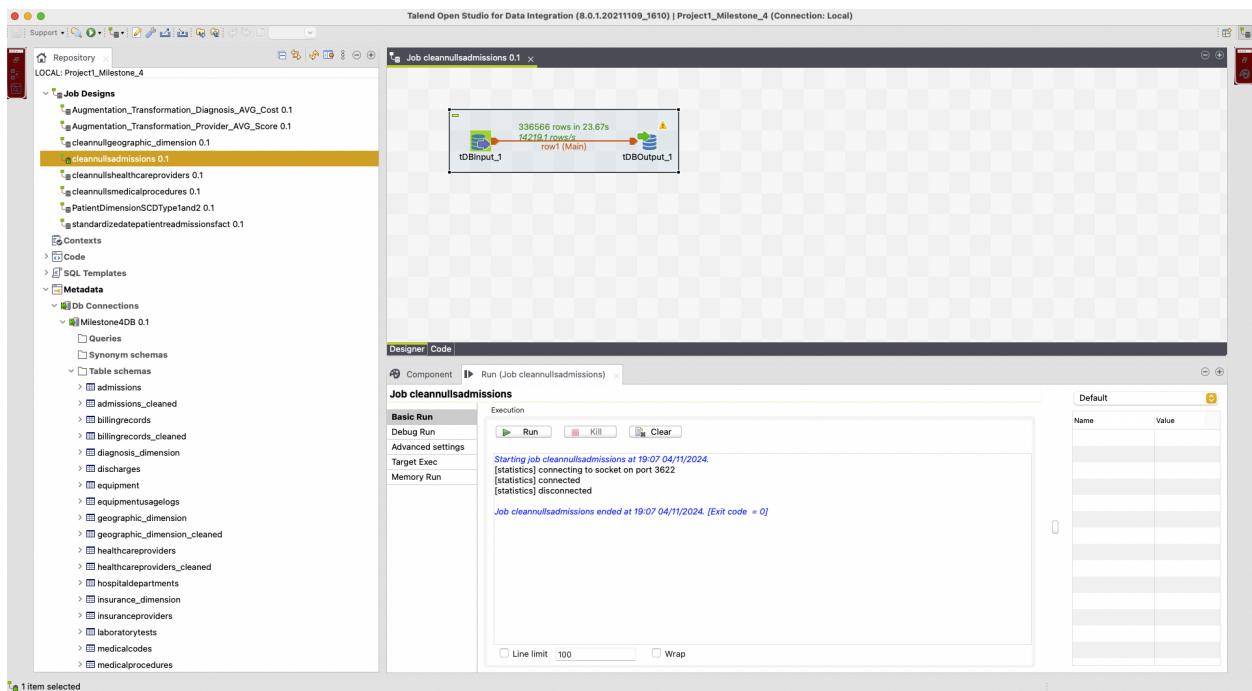


Final Project 1 Report

- Clean NULLs in *Healthcare Providers*

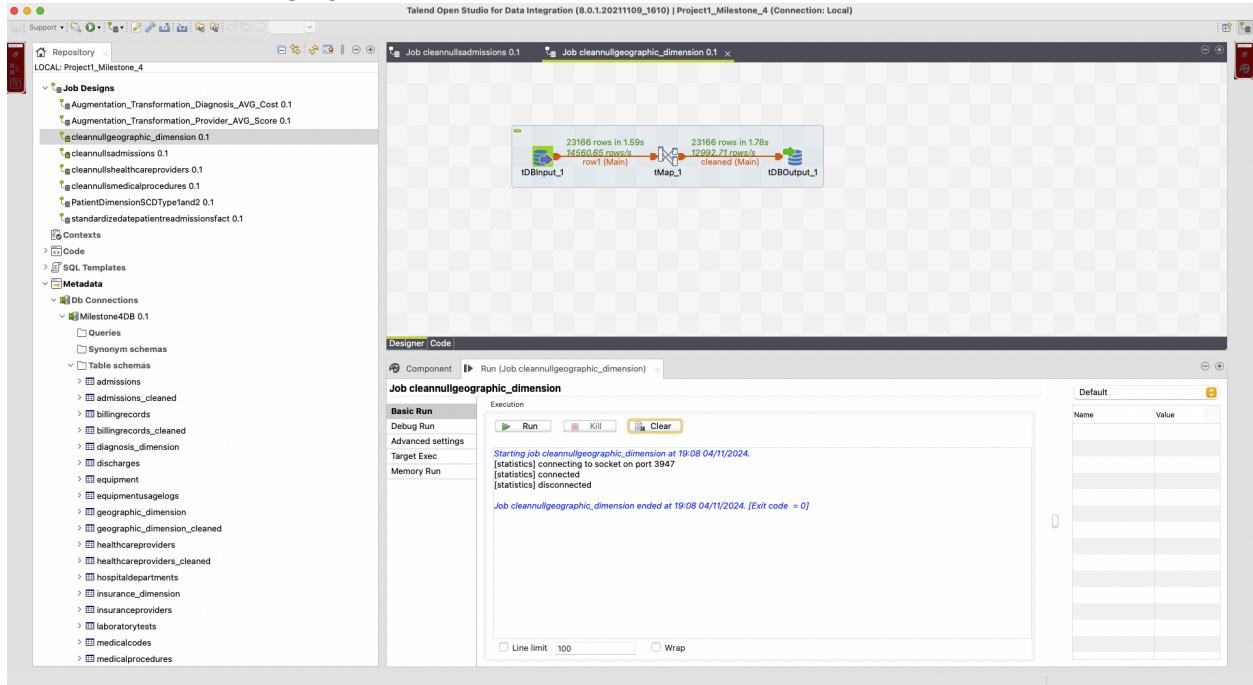


- Clean NULLs in *Admissions*

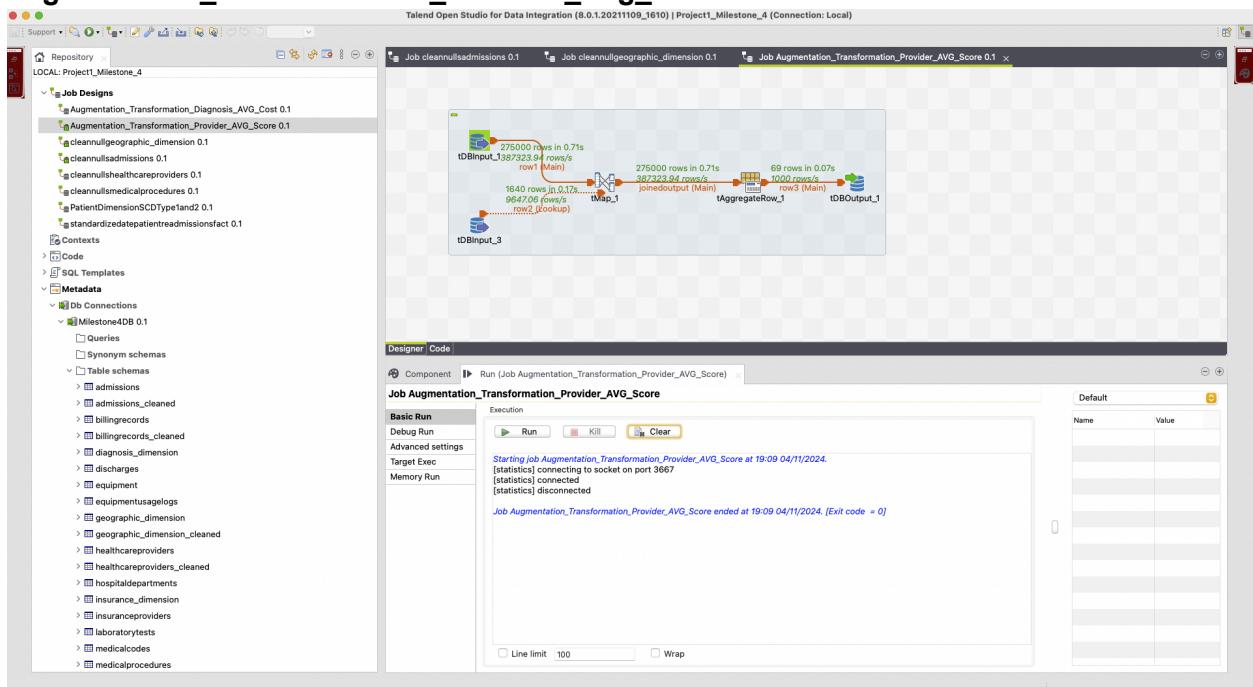


Final Project 1 Report

- Clean NULLs in *Geographic dimensions*

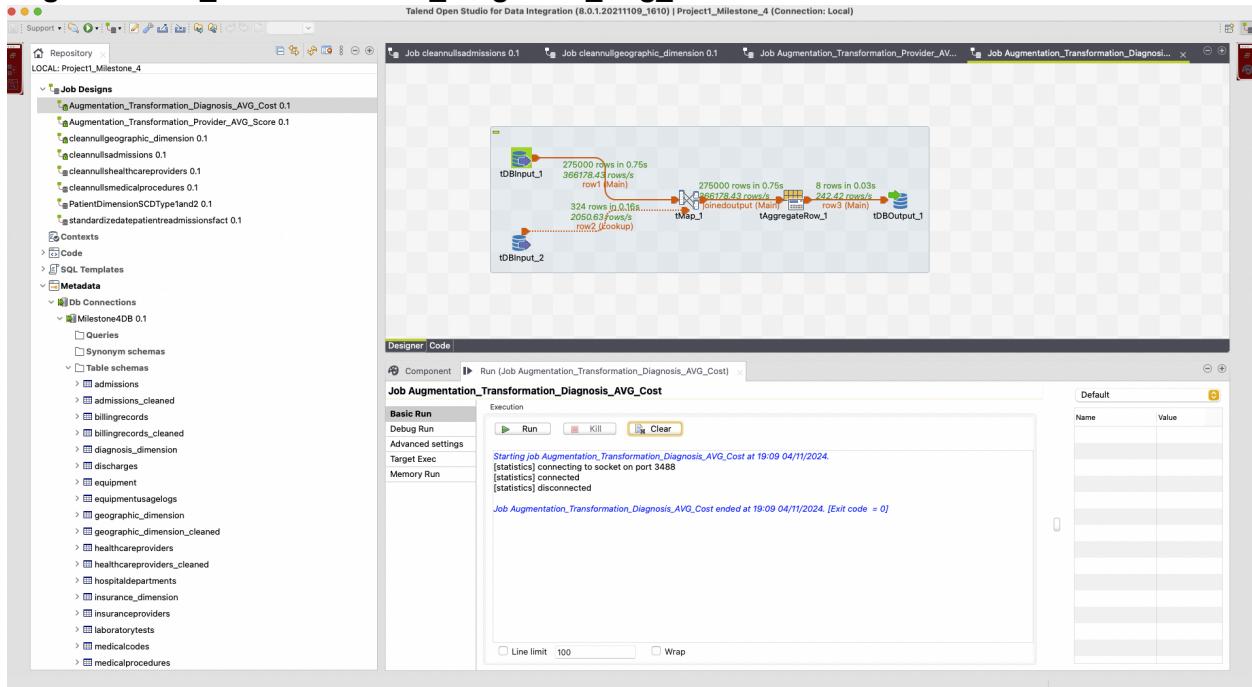


- Augmentation_Transformation_Provider_Avg_Score

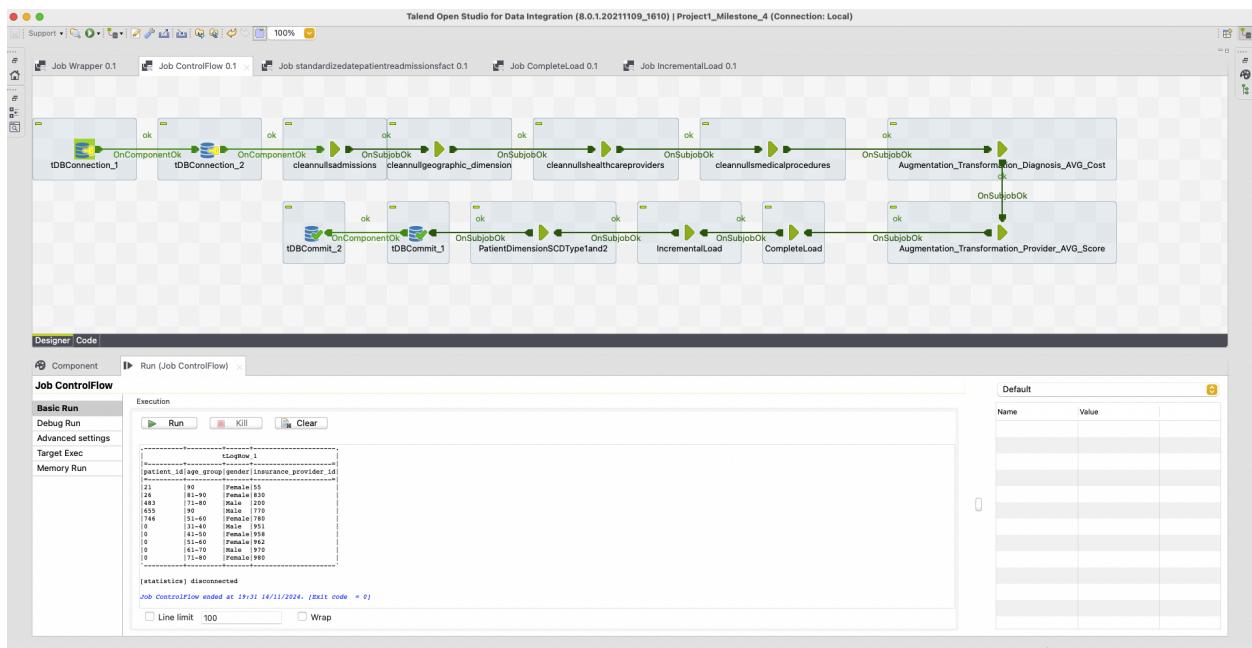


Final Project 1 Report

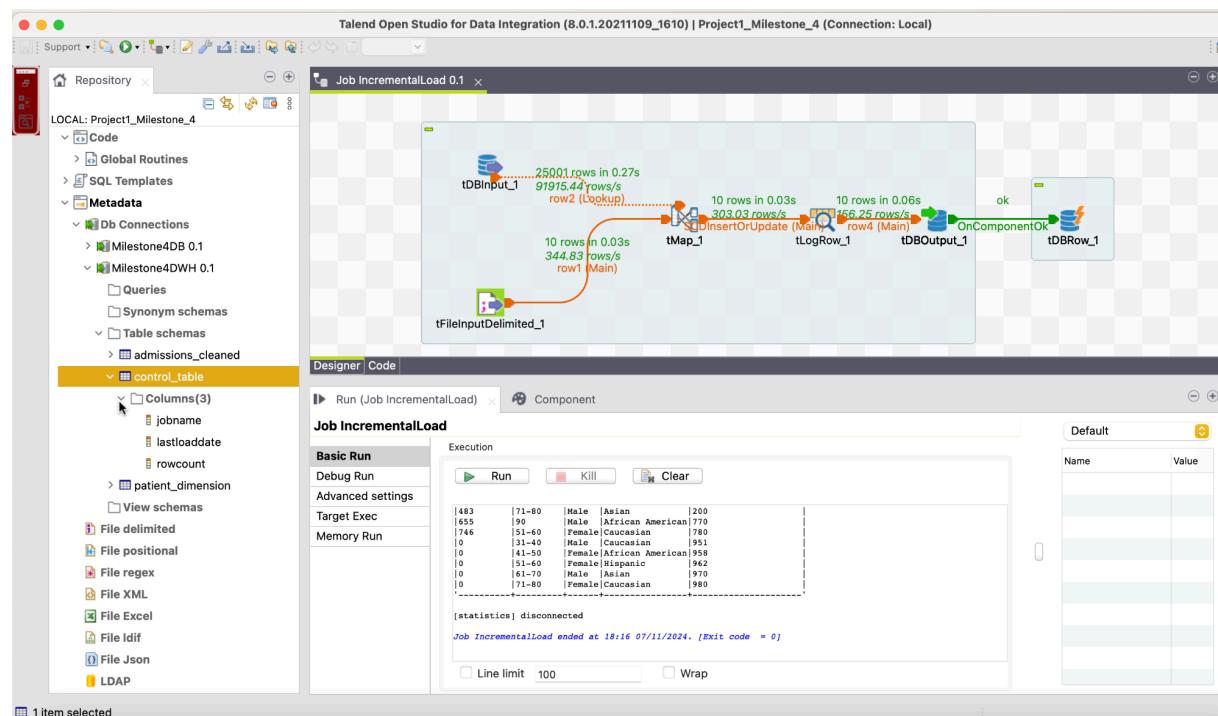
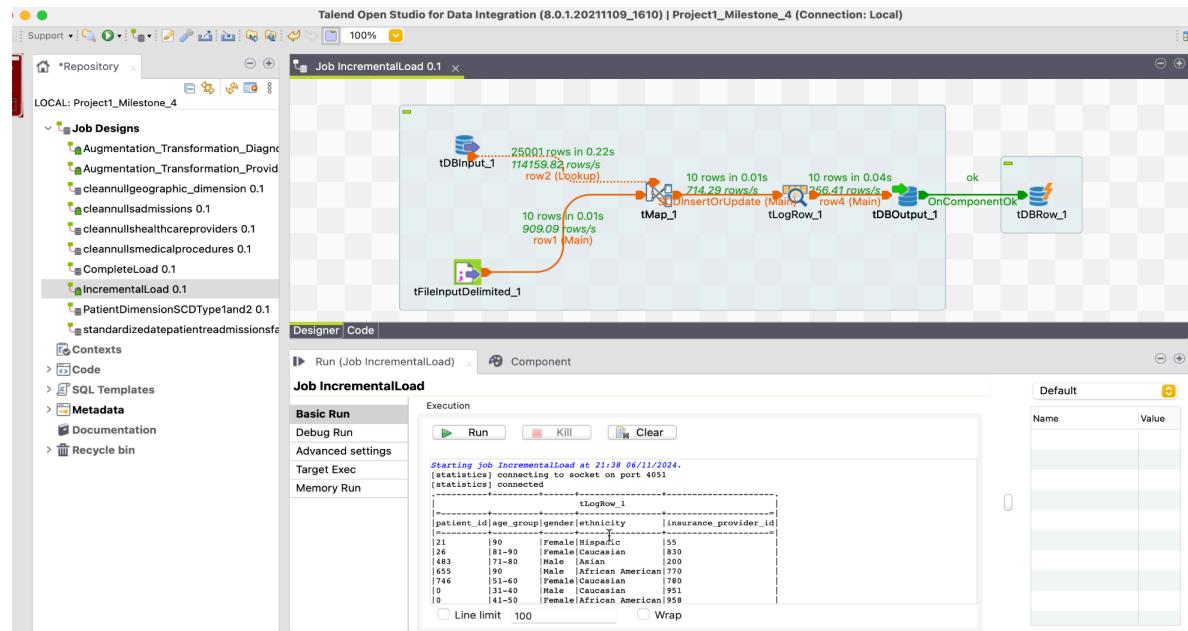
- Augmentation_Transformation_Diagnosis_Avg_Cost



iii. Loads & Control Flow



Final Project 1 Report



Final Project 1 Report

The screenshot shows the pgAdmin 4 interface. The left pane, titled "Object Explorer", displays a tree view of database objects. The "Tables" node under the "data_warehouse" schema has two entries highlighted: "admissions_cleaned" and "patient_dimension". The right pane contains a "Query" editor with the following SQL query:

```
1 Select * from data_Warehouse.control_table
```

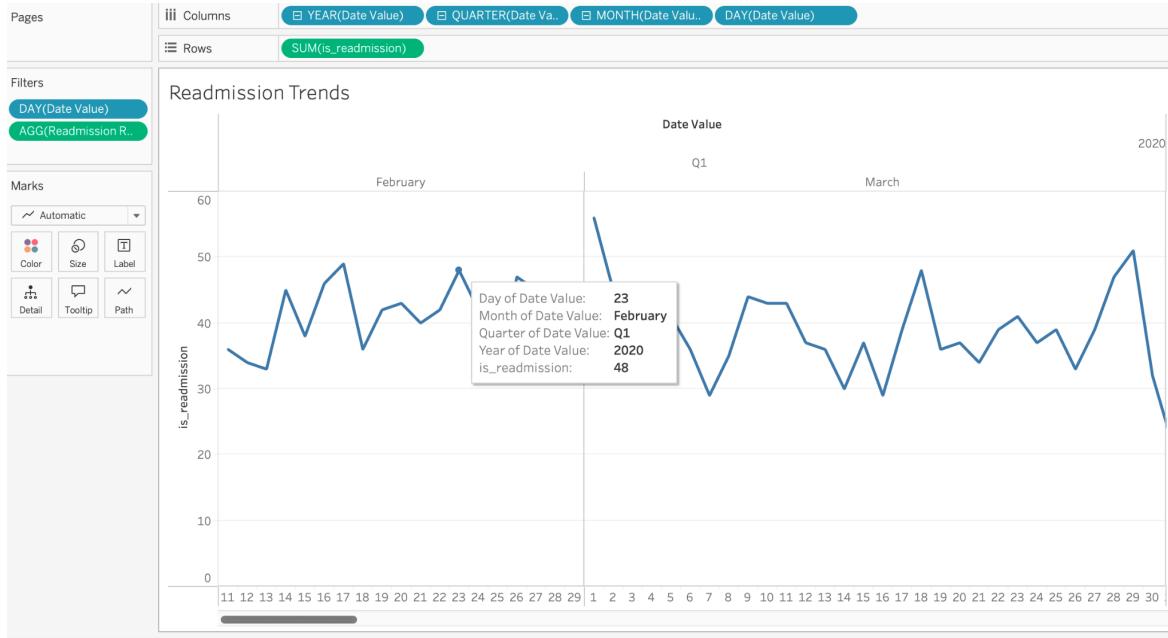
Below the query editor is a "Data Output" tab showing a single row of data from the control table:

jobname	lastloaddate	rowcount
IncrementalLoad	2024-11-06 21:38:47.38597	10

At the bottom of the pgAdmin window, status information is displayed: "Total rows: 1 of 1", "Query complete 00:00:00.139", and "Ln 1, Col 43".

Final Project 1 Report

Readmission Trends: This is one of the KPIs that we used that shows the readmission rates within 30 days. Furthermore, it shows the readmission rates that can be broken down by temporal hierarchy. This shows us



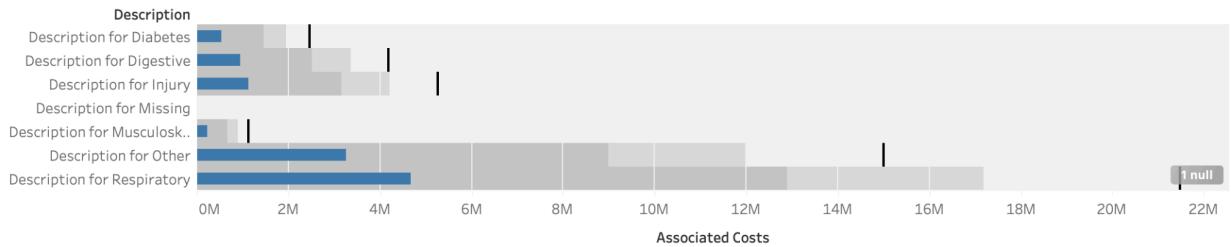
Cost Trends: This KPI shows the cost of readmissions over time.



Cost Analysis: This KPI shows costs per diagnosis.

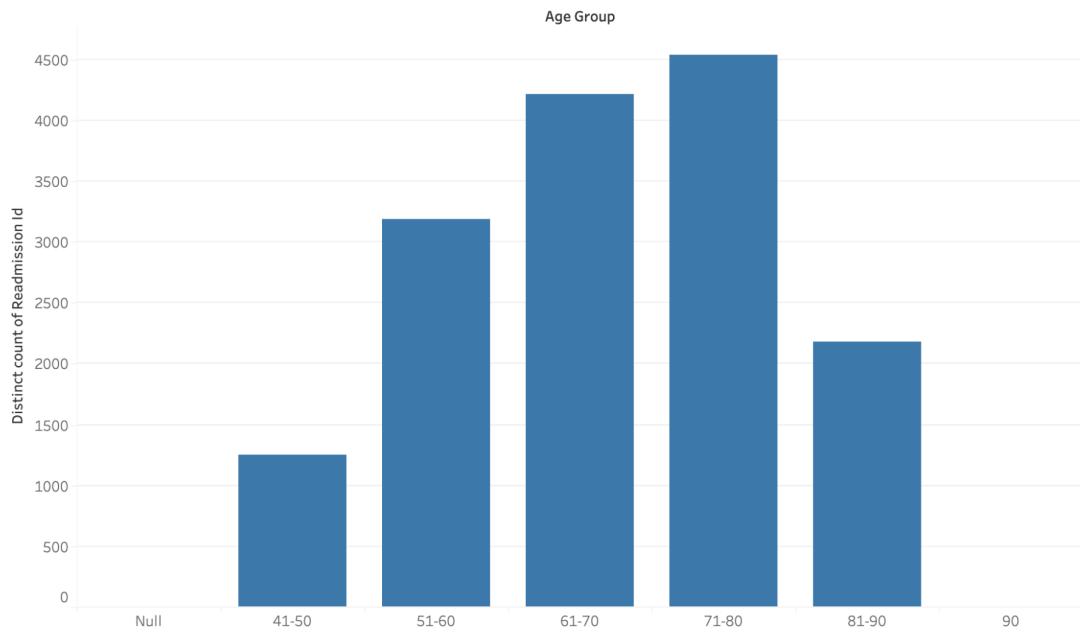
Final Project 1 Report

Cost Analysis



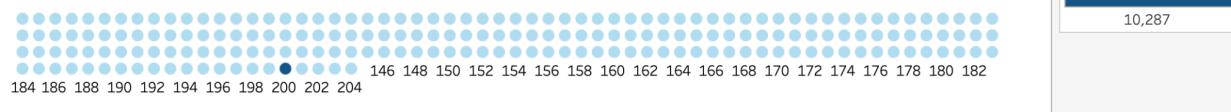
Demographic Analysis: This demographic analysis arranges the number of readmissions per age group.

Demographics Analysis



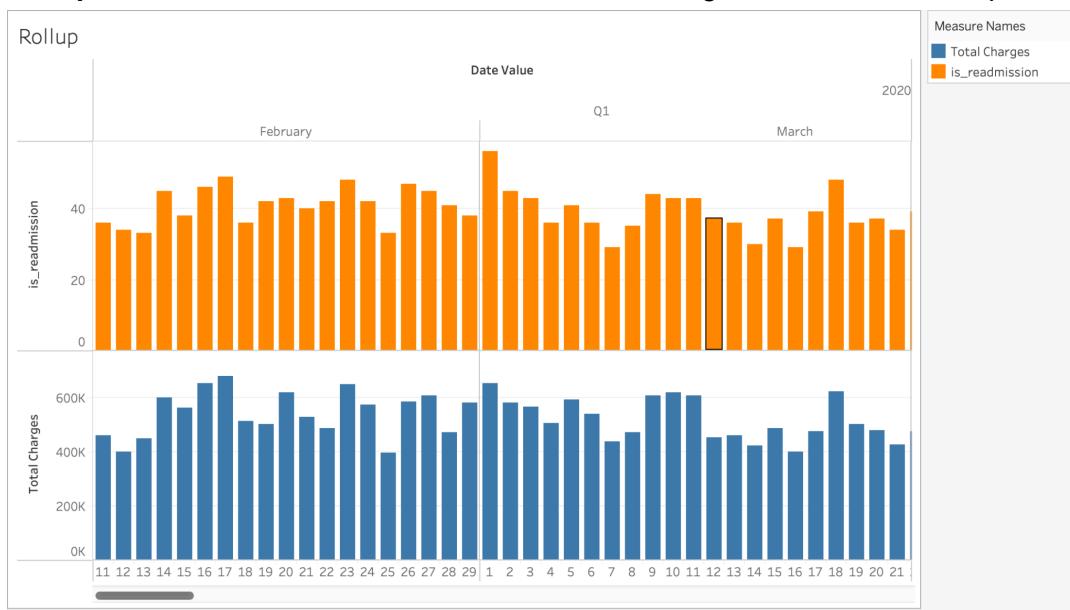
Drill Down - Department

Department/Specialty

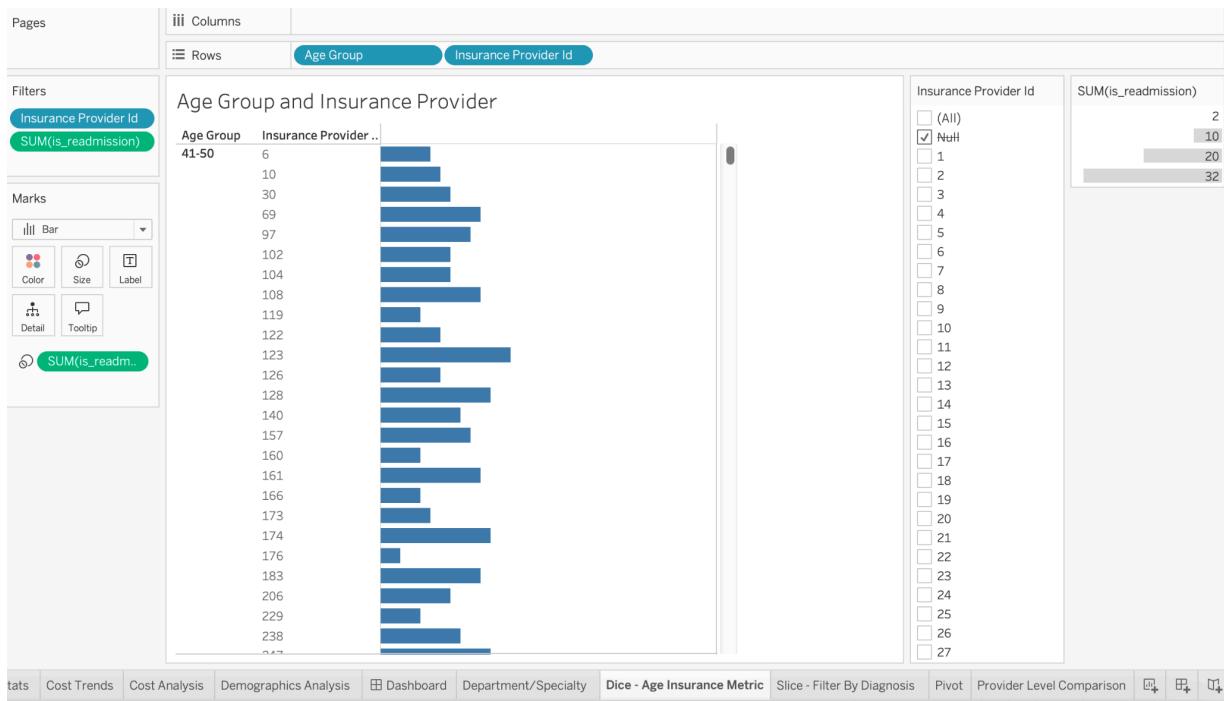


Final Project 1 Report

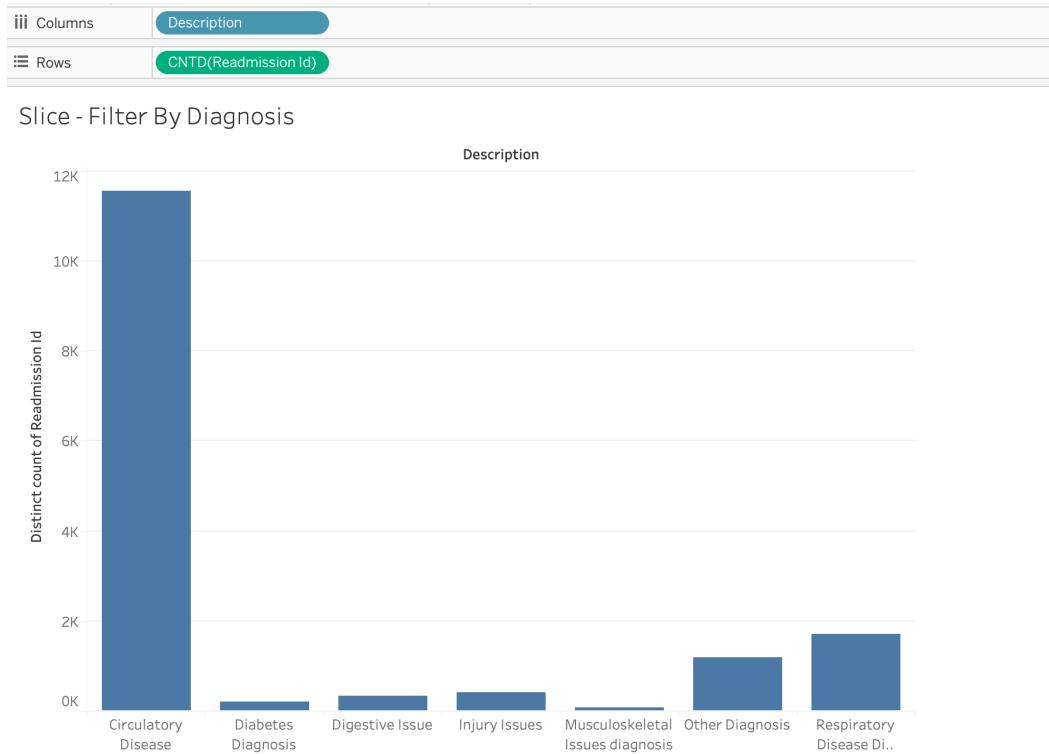
Rollup: This chart shows the admission rate and charges associated with it per day.



Age Group and Insurance Provider: Shows the rate of readmission per age group and insurance provider.



Final Project 1 Report



Pivot: The following chart shows readmission rate by gender and age to identify demographic trends.

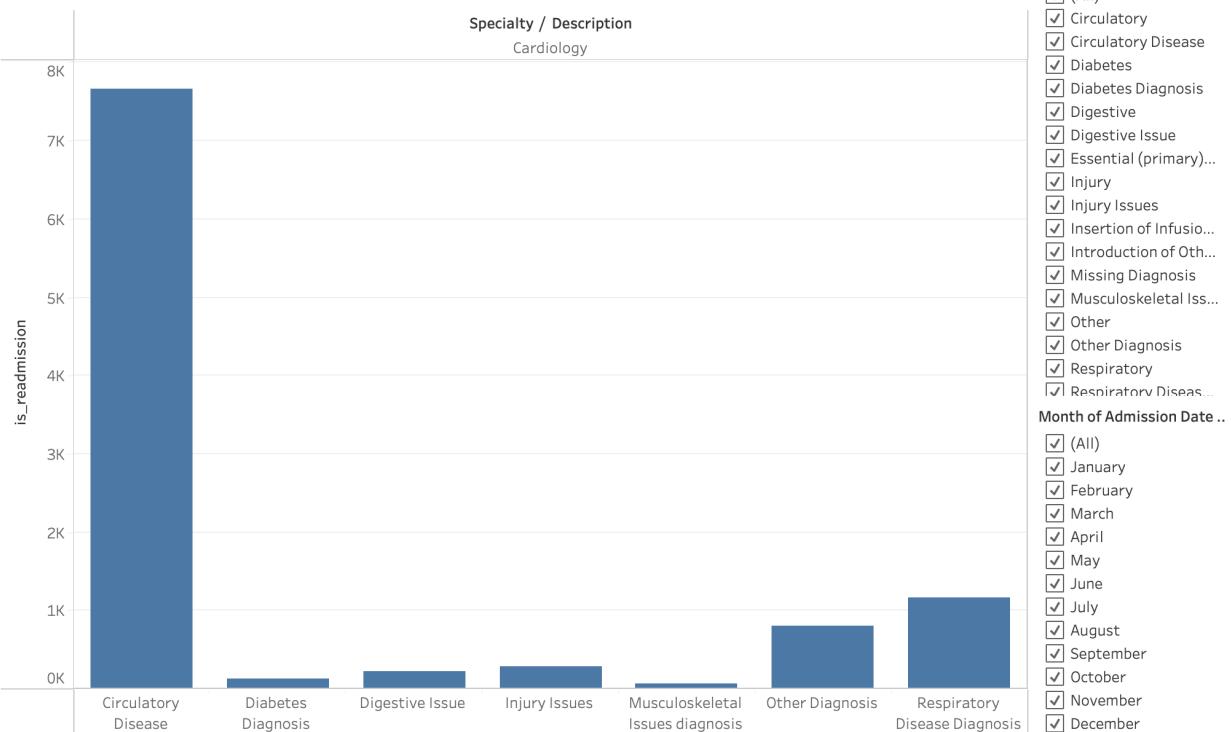
Readmission Rate by Age and Gender

Age Group	Gender	
Null	Null	.
41-50	Female	■
	Male	■
51-60	Female	■
	Male	■
61-70	Female	■
	Male	■
71-80	Female	■
	Male	■
81-90	Female	■
	Male	■
90	Female	.
	Male	.
90+	Female	.
	Male	.

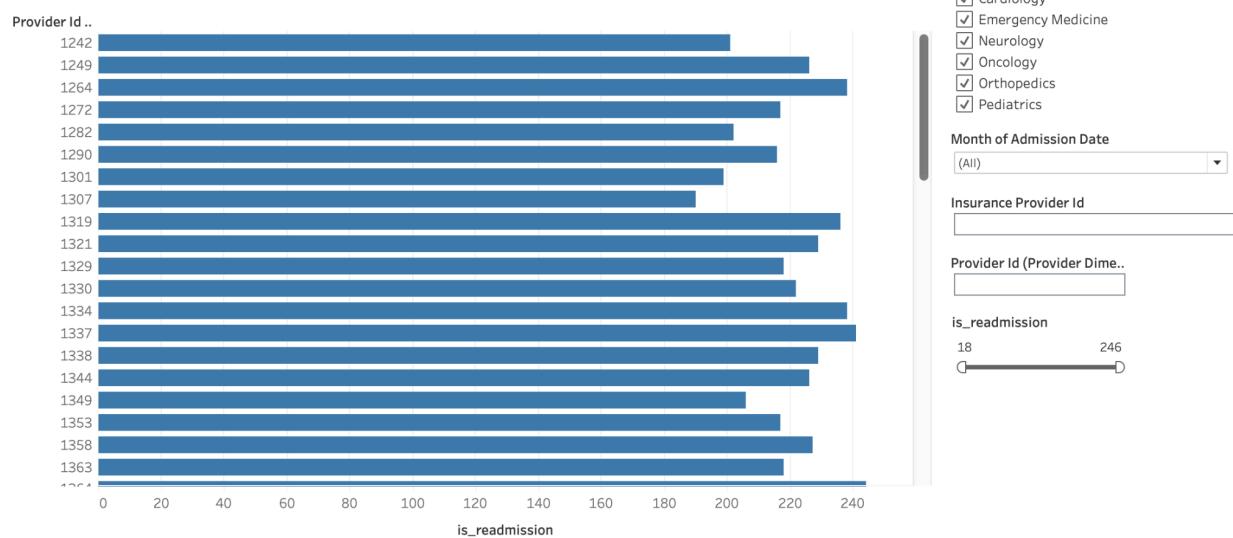
Final Project 1 Report

Drill Across: This chart shows comparison among providers, based on several factors such as their readmission rate, which can be further analyzed month by month. We can also specify a specific number of providers or numbers of readmission. Whereas specialty level summary can drill into readmission rates based on the specialty of the diagnosis. Furthermore, it can do more granular research such as the type of diagnosis and by month.

Specialty Level Summary



Provider-Level Comparison



Final Project 1 Report

Insights:

From these visualizations, some of the key insights we have gained are that, with the number of readmissions, the costs to the hospital also went up. Further, we have noticed that the cardiology department needs an audit, as it was responsible for all of the readmissions. Moreover, we need to provide more care to older patients, as the number of readmissions is significantly higher among older patients.

Link to Tableau File and Presentation: [MILESTONE 7 ON PREM](#)